**UNIVERSIDAD CARLOS III DE MADRID**

**ESCUELA POLITÉCNICA SUPERIOR**

**INGENIERÍA SUPERIOR DE TELECOMUNICACIÓN**

**SISTEMAS Y REDES DE TELECOMUNICACIÓN**


**PROYECTO FIN DE CARRERA**

# IGUALACIÓN DE CANAL NO LINEAL MEDIANTE ALGORITMOS KERNELIZADOS

**Autor: Salvador Ramón Aguilar González**
**Tutor: Angel Navia Vázquez**

**Septiembre de 2009**

Proyecto Fin de Carrera.

Igualación de Canal no Lineal Mediante Algoritmos Kernelizados.

Autor
Salvador Ramón Aguilar González

Tutor
Angel Navia Vazquez

La defensa del presente Proyecto Fin de Carrera se realizó el dia 22 de Septiembre de 2009, siendo calificada por el siguiente tribunal:

PRESIDENTE: Manel Matínez Ramón

SECRETARIO: Emilio Parrado Hernández

VOCAL: Carlos Jesús Bernardos Cano

Y habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

Leganés, a 22 de Septiembre de 2009

A los que me han apoyado y se
han mantenido a mi lado.

## Agradecimientos

Quiero aprovechar estas primeras páginas del Proyecto Fin de Carrera para dar las gracias a mis padres y a mi hermano por el apoyo y la comprensión que me han ofrecido durante toda la carrera. Siempre han estado a mi lado, ayudandome ante las dificultades y disfrutando de las alegrias. Sin ellos no habría llegado a este punto. En particular me gustaría agradecer a mi madre, Maria Luisa, ya que sin su cariño y comprensión hubiera estado perdido, y a mi padre, Salvador, porque sin su tesón y perseverancia, me hubiera equivocado de camino. Quería agradecer igualmente a mi hermano, Francisco, por ser un ejemplo que seguir y un modelo al que llegar. Mi familia me ha marcado la dirección que culmina con la defensa del presente Proyecto Fin de Carrera.

Las siguientes palabras, son para agradecer a mi novia, Sandra, que haya estado a mi lado durante la realización de este proyecto. Soy afortunado, ya que sin su apoyo, amor y paciencia hubiera resultado muy difícil la culminación de este trabajo.

Durante la carrera he encontrado multitud de amigos, que me acompañaran toda mi vida. Me refiero a Omar, Luis, Cesar, Diego, Blanca, Elvira, etc. Su apoyo y su ayuda han sido muy importantes para mí. Tengo que darles las gracias por su amistad, comprensión, confianza y sobre todo por los buenos recuerdos durante estos años.

Quiero agradecer de manera especial a mi tutor, Angel Navia, la ayuda, la atención y el trato que me ha brindado durante la realización del Proyecto Fin de Carrera. En particular, me gustaría darle las gracias por su gran paciencia, dedicación y confianza sin la que este proyecto nunca podría haberse hecho realidad.

Me gustaría igualmente agradecer a la gran cantidad de compañeros que me han rodeado durante innumerables horas de clase y prácticas. Hemos aprendido y crecido mucho juntos. Ha sido una suerte y un honor compartir esta etapa de mi vida con vosotros.

Por último, me gustaría agradecer a los autores del algoritmo KRLS, Y. Engel, S. Mannor y R. Meir, ya que su excelente trabajo me ha podido servir de base para la realización del Proyecto Fin de Carrera.

## Resumen

En el presente Proyecto Fin de Carrera se aborda un estudio sobre la problemática de la igualación de canal bajo entornos no lineales variantes. Se examina el comportamiento de la versión kernelizada del algoritmo RLS, denominada KRLS, bajo un entorno de igualación de las citadas características. Para comprobar su comportamiento se desarrollan experimentos incrementales en complejidad en un marco de comparación con otros algoritmos basados en el estado del arte del problema: Redes neuronales, series de Volterra, algoritmos clasificadores, etc. . Dichos experimentos parten de canales estáticos, para acabar con canales dinámicos, aplicando tanto un entrenamiento con conocimiento de los datos de entrada, como un entrenamiento guiado por decisión.

# Índice General

# Lista de Figuras

# Lista de tablas

# Capítulo 1

## *1.1 Introducción a la igualación adaptativa de canal*

En este primer capítulo del Proyecto Fin de Carrera, vamos a intentar dar una visión global del sentido y objetivo del proyecto en sí. Además queremos aprovechar éste apartado para comentar la estructura del documento, así como la relación existente entre los capítulos.

El objetivo del proyecto, es el análisis de las prestaciones de un algoritmo mediante la comparación con otros semejantes bajo las mismas condiciones de ejecución. El algoritmo bajo estudio se denomina KRLS (Kernel Recursive Least Squares) y su implementación será la que usaremos para comparar con las implementaciones de otros algoritmos de referencia. En particular, vamos a plantear un caso de estudio particular, para nuestros algoritmos, el problema de igualación de canal, que comentaremos más adelante, y en detalle la igualación de canal no lineal y con canal variante en el tiempo. Tal y como hemos mencionado, en este punto no queremos introducirnos al detalle en cómo hemos realizado nuestro trabajo, ya que eso lo comentaremos más adelante.

Para comentar brevemente como vamos a organizar el documento, tenemos como primer punto el apartado en el que nos encontramos, que viene a ser un simple brochazo inicial sobre el que construiremos todo. El segundo apartado, será la base teórica, sobre la cual levantaremos los pilares explicando los conceptos necesarios para la comprensión de los resultados. El tercer apartado corresponde con el estado del arte, en el que explicaremos más en detalle nuestro problema en particular, justificando nuestro estudio sobre las tecnologías que se están implementando en la actualidad. Con estos puntos,

tendremos un marco completo de la teoría de nuestro problema de igualación, así como los avances que se desarrollan actualmente y entre los cuales se encuentra el algoritmo KRLS que será el centro de nuestro trabajo.

Dejando los puntos teóricos a un lado, tendremos seguidamente el punto de resultados experimentales, en el cual comentaremos con detalle el resultado de nuestro estudio. Por último, tendremos el capítulo de conclusiones que pretenderá finalizar todo lo dicho, ofreciendo a modo de resumen las ideas básicas de nuestro proyecto.

Los siguientes puntos del documento corresponderán con la bibliografía y los apéndices a los cuales se hace referencia durante el desarrollo del proyecto.

## *1.2 Información*

El ser humano es un ser social y necesita comunicarse, necesita manejar información, modificarla y transmitirla, así como recibirla del entorno y de los demás seres humanos. Nuestro propio diseño está preparado para obtener esa información del entorno, ejemplo de ello son nuestros sentidos.

Los sentidos toman la información de la naturaleza que nos rodea, la manipulan y la hacen llegar al cerebro donde es interpretada. Esta información es la forma en que nuestros sentidos pueden plasmar el entorno que nos rodea para poder procesarlo. De esta manera cuanto más información sea posible obtener y procesar, mas probabilidades tendrá el sujeto en cuestión de poder sobrevivir en un entorno hostil. Los sentidos son en este caso la interfaz que permite comunicar nuestro entorno y nuestro cerebro, mediante la toma de información.

Además de nuestra propia estructura de sistema de información, En nuestros tiempos, se está produciendo una nueva revolución en el significado de información, el cual se está poniendo aún mas en evidencia con la acuñación de términos como TIC (Tecnologías de la información y de la comunicación) y sociedad del conocimiento y sociedad de información.

Podemos entender sociedad de la información como una sociedad en la que se la creación, procesamiento, manipulación y distribución de la información son parte de las actividades culturales y económicas de la propia sociedad. El origen de estas sociedades viene determinado por el uso de las TIC, que son las tecnologías que permiten el estudio, desarrollo, implementación, almacenamiento y distribución de la información. El nacimiento de estas sociedades está cumpliendo un hito en nuestra historia, pero el siguiente paso es poder comprender esa avalancha de información, creando de esa manera ideal el último termino, ‘sociedad del conocimiento’.

Esto supone un paso mas sobre los sistemas de toma de información que forman nuestros sentidos, Con estas premisas, tenemos una sociedad en la que prima la comunicación, la información y en la que esta, irá teniendo mayor peso con el tiempo.

La comunicación de información nos hizo evolucionar y estamos evolucionando hacia el acceso a la información universal.

## *1.3 Tratamiento de información*

La importancia de la extracción de información es indudable, ya que sin ella no habría base sobre la que trabajar. No obstante, la información puede variar, no siendo el mecanismo empleado para obtener la información el mas adecuado, ya que de algún otro modo obtendríamos unos datos mucho mas valiosos. Este efecto se conoce como variabilidad de la información o el entorno y hace que se desarrollen tecnologías que se adaptan a dicha variación. Como ejemplo de ello en nuestro propio cuerpo, se encuentra el sentido de la vista, que dilata o contrae las pupilas para manejar la intensidad de la luz que inciden en las células nerviosas que se encuentran en la retina, o en el sentido del tacto, el movimiento de separación ante un objeto con demasiada intensidad, como el fuego.

La variabilidad del entorno, así como la adaptación de los seres a ese entorno variable está determinada directamente con la adaptación del entorno por parte del ente que toma los datos de interés.

Al proceso comentado de extraer información sobre un determinado espacio de interés de un entorno ruidoso, le vamos a denominar a partir de ahora con el termino ‘filtrado’, englobando a la variante adaptativa como ‘filtrado adaptativo’ [Haykin, 2002].

Podemos usar un filtro para realizar algunas operaciones básicas de procesamiento de información:

- Filtrado: La extracción de información sobre el campo de interés en el instante actual se realiza usando los datos obtenidos en el instante inicial, así como en instantes anteriores.
- Suavizado: La extracción de información para el instante actual no se necesita en el mismo instante, por lo que la obtención de información se basa en los datos obtenidos en el instante actual, y en instantes anteriores y posteriores. De esta manera incurrimos en un retardo.
- Predicción: El objetivo es intuir información futura sobre el campo de interés usando los datos obtenidos en el instante actual y en anteriores.

### 1.3.1 Tipos de problemas

Hemos comentado las operaciones básicas que podemos realizar con filtrado, pero si observamos detenidamente estas tareas, podemos agruparlas en dos clases aún más básicas. De esta manera reduciríamos los problemas de filtrado y suavizado a un mismo padre que denominaremos 'decisión', ya que en definitiva el problema es obtener unos resultados de un campo deseado a partir de unos datos de partida ruidosos que contienen a los datos deseados. Por otra parte mantendremos el segundo grupo como 'estimación'.

La fuente de datos de estos dos grandes grupos proviene de fuentes ruidosas en las cuales se encuentran los datos de interés. La principal diferencia entre ambos grupos es la información que queremos obtener: para el caso de la estimación tenemos que la información que deseamos consiste en el valor de una o varias magnitudes continuas, mientras que en el caso de la decisión, nuestra salida será una hipótesis que consideraremos correcta de dentro de un conjunto finito de ellas, es decir nuestra región de información de salida es discreta. Podemos ver la diferencia entre ambos procesos en la Figura 1.1 [TDS, 2005].

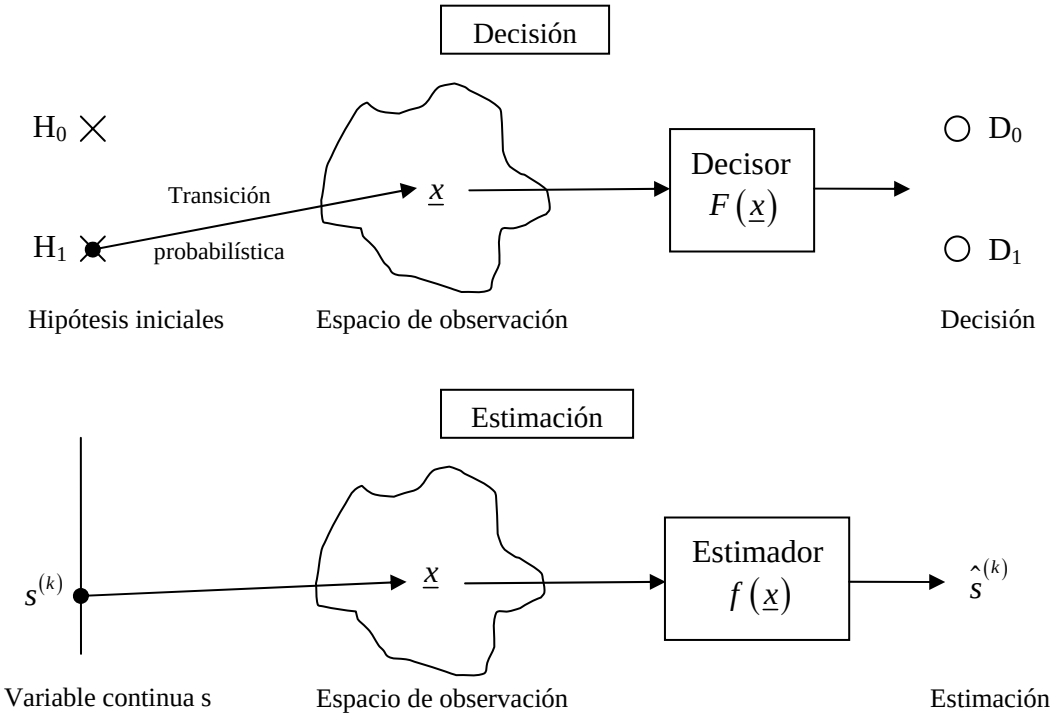



**Figura 1.1 – Procesos de decisión y estimación.**

Una vez comentados los diseños básicos de nuestro problema, tenemos a su vez diferentes metodologías de implementación de la solución de nuestro problema. Las comentamos en la Tabla 1.1 (Bishop, 1995; Hayes, 1996; Duda, 2001]).

| **Aproximación empleada** | |
|---|---|
| Máquina | Se emplea esta aproximación cuando se dispone de un conjunto de datos y se desconoce la naturaleza del problema. Se elige una arquitectura con suficiente capacidad expresiva y se entrena con nuestro conjunto de datos de ejemplo para que sea capaz de resolver el problema. |
| Analítica | Cuando tenemos un conocimiento de la 'física' del problema, disponemos de información estadística precisa mediante la cual, minimizando un coste, podemos obtener el diseño completo y óptimo del estimador/decisor, incluyendo la arquitectura y los parámetros. |
| Semianalítica | No disponemos de la información estadística completa, así que estimamos a partir de las observaciones la información analítica que falta y se realiza un diseño analítico a partir de estas estimaciones. |

**Tabla 1.1 – Aproximaciones empleadas en el diseño de estimadores/decisores.**

También hemos de comentar los inconvenientes de cada aproximación:

- Máquina: Principalmente tenemos la dificultad de elegir el modelo adecuado, así como el número de parámetros que emplearemos. Si esta elección no es acertada, incurriremos en fenómenos no deseados como:

    - Subajuste: La máquina presenta incapacidad expresiva o deficiencia en su entrenamiento que imposibilita encontrar una solución que represente apropiadamente la relación existente entre el espacio de entrada y salida.
    - Sobreajuste: En este caso la máquina dispone de una elevada capacidad expresiva que produce una solución demasiado específica del conjunto de entrenamiento. En este caso, no tenemos generalización, y obtendremos resultados malos en conjuntos de entrada que no sean semejantes al de entrada.

Las ventajas que encontramos en esta aproximación son que se apoyan sobre datos reales y que no presupone ningún modelo, además todo se dirige a resolver el problema porque todo está orientado al problema. Como desventajas podemos destacar que el método necesita información

representativa del problema, sino el entrenamiento no es efectivo. Además es complicado elegir la arquitectura, así como los parámetros necesarios.

- Analíticos: Son diseños óptimos y directos al problema en cuestión, y como ventaja frente a los diseños máquina tenemos que en este caso la máquina[1], la da el propio procedimiento. Como desventajas tenemos que son poco robustos, ya que si se equivoca no sirve para nada o si cambian las condiciones de funcionamiento el sistema se degrada.
- Semianalíticos: Este diseño no permite llegar a resultados óptimos, ya que aunque hagamos las cosas razonablemente bien, siempre obtendremos resultados subóptimos. Al igual que en el diseño máquina, tenemos el inconveniente de que los datos que utilicemos para estimar la información estadística, deben de ser representativos del problema. Además su robustez depende muy directamente de los estimadores en los que se base.

[1] Entendemos máquina como el conjunto formado por la arquitectura y por los parámetros.

## *1.4 Filtrado de señal*

Un sistema puede ser visto como cualquier proceso que resulta en la transformación de señales, teniendo una señal de entrada y una de salida la cual está relacionada con la entrada a través de la transformación del sistema [Oppenheim, 1996]. Enlazando esta definición de sistema genérico podemos determinar otra, sobre el concepto de filtro como un sistema automático artificial, que produce una o varias salidas como resultado de una serie de entradas. En general los sistemas suelen ser lineales e invariantes en el tiempo (LTI). En este caso podemos determinar a un sistema mediante su respuesta al impulso, siendo el caso en el que en la entrada del sistema introducimos una señal delta[2].

En general para la definición de filtros y de sistemas en general, podemos clasificarlos en IIR[3] (duración de respuesta a impulso infinita) o FIR (duración de respuesta al impulso finita). La diferencia entre ambos modelos es que los FIR tienen una memoria finita, mientras que los IIR tienen memoria infinita. En el caso particular de los IIR tenemos la problemática de su inestabilidad ya que son sistemas realimentados, por lo que hay que elegir sus parámetros con bastante exactitud.

Otro factor a tener en cuenta en la definición de filtros, es si van a trabajar en tiempo continuo o discreto. En cualquier caso, un sistema continuo puede ser empleado para tiempo continuo y viceversa realizando un muestreo de la señal continua[4] o un interpolado en la señal discreta

[2] Se suele denominar como función delta de Dirac en el caso continuo, como delta de Kronecker en el caso de tiempo discreto.
[3] IIR: Infinite Impulse Response ; FIR: Finite Impulse Response.
[4] Este proceso de muestreo debe realizarse de acuerdo a ciertas restricciones con el fin de garantizar que no hay pérdida de información [Nyquist, 1928; Shannon, 1949].

### 1.4.1 Filtrado lineal vs. filtrado no lineal

En función de las características del problema debemos elegir la estructura de filtrado entre lineal y no lineal. Es una elección primordial, ya que los resultados obtenidos, así como la complejidad del modelo dependerán de ello directamente.

Podemos determinar que un filtro es lineal si la salida del dispositivo es una función lineal de las observaciones aplicadas a la entrada del filtro. En el caso contrario, tendremos un filtro no lineal [Haykin, 2002].

Normalmente se suelen elegir los filtros lineales, ya que suponen una menor complejidad, aun permitiendo un cierto grado de error frente a la alternativa no lineal. Por el contrario, en determinados escenarios el uso del filtrado lineal no tiene cabida, tales como igualación en el caso de canales no lineales, o en el caso de análisis de señales de voz, en las que el uso de filtrado no lineal supone una gran ventaja contra su opción lineal.

Un ejemplo de filtrado lineal junto con aproximación analítica (asumimos que conocemos ciertos parámetros estadísticos) es aquel en el que tenemos una señal deseada junto con ruido aditivo no deseado y deseamos diseñar un filtro que nos permita tomar esos datos ruidosos como entrada y nos ofrezca una salida en la que minimicemos los efectos del ruido mediante un determinado criterio estadístico.

Es usado normalmente como criterio estadístico en este tipo de problemas de optimización de filtros, el mínimo error cuadrático medio, entendiendo el error como la señal diferencia entre la señal deseada y la salida real del filtro.

Una vez comentado uno de los marcos de problema global de filtrado lineal, podemos añadir el caso de que la señal de entrada sea estacionaria[5]. Para esta hipótesis de situación, tenemos una solución muy conocida y denominada como filtro de Wiener [Wiener, 1949] el cual da la solución óptima en sentido de error cuadrático medio.

[5] Entendemos que una señal es estacionaria cuando sus características estadísticas en promedio en tiempo son constantes, pudiendo ser la señal determinista o no.

No obstante, tenemos que darnos cuenta que nos hallamos sobre unas hipótesis muy específicas como son la estacionariedad de la señal de entrada. Con tal motivo otra solución muy conocida y denominada filtro de Kalman [Kalman, 1960] en la que asumimos una forma variante en tiempo, o lo que es lo mismo, podemos aplicarla en situaciones de no estacionariedad.

### 1.4.2 Filtrado adaptativo

Entendemos el filtrado adaptativo como un filtro capaz de modificar sus parámetros de forma que logre minimizar una función de coste. Generalmente consideramos esa función de coste basada en el error cuadrático medio como la diferencia entre una salida deseada $d(n)$[6] y la salida real del filtro $y(n)$. Se elige este tipo de función de coste, ya que tiene una superficie de función de error convexa, con lo que presenta un único mínimo global, evitando de esta forma caer en mínimos locales durante el proceso de optimización del filtro. El filtrado adaptativo lo podemos ubicar entre las aproximaciones máquina y semianalítica, los cuales nos ofrecían una solución cercana a la óptima.

En el mismo momento en el que mencionamos filtrado adaptativo, tenemos que tener en cuenta que estamos refiriéndonos a algoritmos basados en recursión, ya que basamos el aprendizaje de nuestro filtro en el error que obtenemos en cada instante. Esto hace posible al filtro trabajar exitosamente en entornos en los que el completo conocimiento de las características de la señal deseada no está disponible.

De la manera comentada, tenemos que el algoritmo adaptativo, bajo la hipótesis de encontrarnos en un entorno estacionario, después de un numero de iteraciones convergerá a la solución de óptima de Wiener [Wiener, 1949]. Si por el contrario, suponemos encontrarnos en un entorno no estacionario, tenemos que un algoritmo adaptativo ofrece capacidad de seguimiento, o lo que es lo mismo, podemos seguir las variaciones que tienen las características estadísticas de la señal de entrada en el tiempo, siempre y cuando la variación sea lo suficientemente lenta.

Derivado de lo comentado anteriormente, tenemos que el algoritmo adaptativo es un algoritmo recursivo que actualiza sus parámetros en cada iteración, con lo que nos encontramos en la situación en la que el propio algoritmo es dependiente de los datos con los cuales le entrenamos. Con esto podemos decir que, el filtrado adaptativo es realmente un elemento no lineal en el sentido de

[6] Usamos $d(n)$ en lugar de $d(t)$, ya que suponemos que nos encontramos en un caso discreto en vez de uno continuo. Siendo en cualquier caso ‘n’ la variable tiempo en discreto y ‘t’ en continuo.

que no obedece el principio de superposición[7]. No obstante obviando este hecho, podemos englobar el filtrado adaptativo en dos grandes grupos: lineal y no lineal. En el primer caso nos encontramos cuando la salida estimada de nuestro filtro es calculada como combinación lineal de las observaciones de entrada, y nos encontraremos en el segundo caso si esta premisa no se cumple.

Para comentar de manera mas precisa el funcionamiento del filtrado adaptativo, vamos a examinar cada uno de sus pasos:

- Filtrado: Mediante este proceso obtenemos la salida $y(n)$, en respuesta de la secuencia de datos de entrada, $u(n)$. Para representar este usamos un esquema transversal[8] como se puede ver en la Figura 1.2[9]

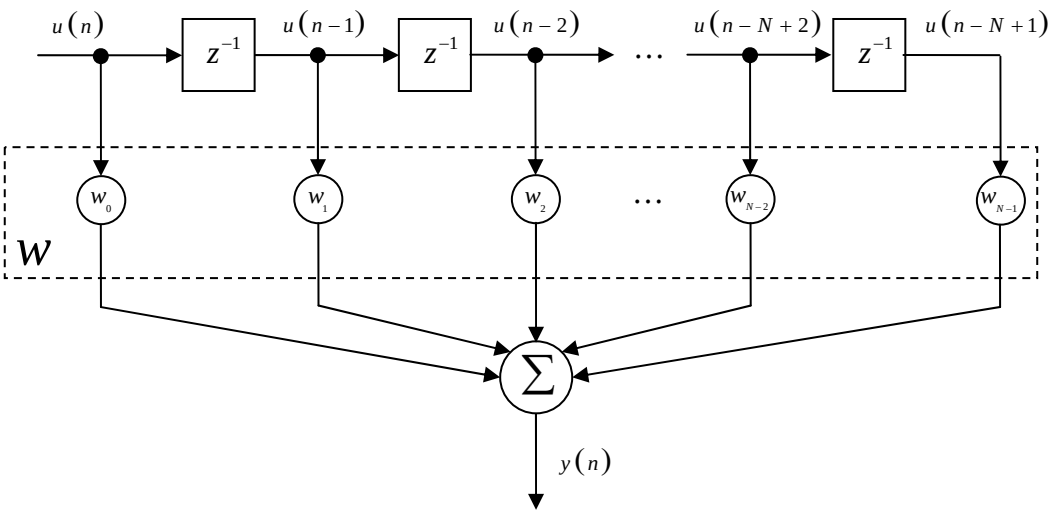



**Figura 1.2 – Filtro transversal**

En el caso comentado, tenemos que el valor N corresponde con la longitud del filtro, o lo que es lo mismo, el número de retardos que se emplean y que determinan la duración de la respuesta al impulso del sistema, así como el número de elementos que se combinan linealmente para producir la salida del filtro.

[7] Entendemos el principio de superposición desde el punto de vista matemático como la división de un problema lineal en subproblemas más sencillos con el objetivo de obtener el problema inicial como suma de los subproblemas sencillos.
[8] Propuesto inicialmente por Kalman [Kalman, 1940].
[9] Usamos el esquema transversal por sus ventajas de implementación y estabilidad. Otras posibles opciones con la red en celosía o el array sistólico [Haykin, 2002].

De esta forma, la salida del filtro la podemos especificar de la forma que muestra la expresión 1.1 y que recibe el nombre de suma de convolución, donde $\underline{w} = \left[ w_0, w_{1, \dots,} w_{N-1} \right]^T$ corresponde con los coeficientes del filtro y $\underline{u}(n) = \left[ u(n), u(n-1), \dots, u(n-N+1) \right]^T$ representa las entradas del filtro.

$$y(n) = \underline{w}^{\mathrm{T}} \underline{u}(n) = \sum_{k=0}^{N-1} w_k u(n-k) \qquad \textbf{1.1}$$

- Adaptación: Con esta etapa proporcionamos un mecanismo para el control adaptativo del conjunto de parámetros configurables del filtro. Este proceso se basa en las muestras una a una según van llegando con lo que reducimos la memoria necesaria. El criterio en el que nos basamos para modificar el comportamiento del filtro suele ser el error que encontramos a la salida. En la ecuación 1.2 podemos ver como se modifica el comportamiento del filtro a través de la modificación de su vector de coeficientes $\underline{w}$.

$$\underline{w}(n+1) = \underline{w}(n) + f\left(\underline{w}(n), e(n), \underline{p}(n)\right) \qquad \textbf{1.2}$$

En la expresión 1.2 tenemos que la actualización de los coeficientes del filtro depende de el propio valor de los coeficientes en el instante inicial, del error definido como la diferencia entre la salida esperada y la salida ideal de nuestro filtro, tal y como muestra la Figura 1.3, y $\underline{p}(n)$ entendido como el estado en el que se encuentra el filtro. La diferencia entre los diferentes algoritmos adaptativos radica en la diferente construcción del vector de estado $\underline{p}(n)$, y en la función de actualización $f(\bullet)$ que se emplee en cada caso.

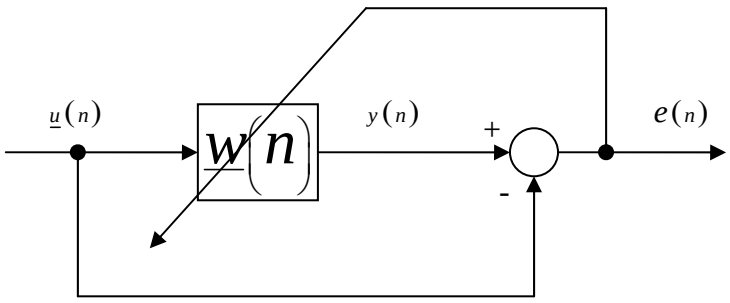



**Figura 1.3 – Esquema de filtrado adaptativo**

#### 1.4.2.1 Elección correcta del algoritmo adaptativo

Una vez comentados los pormenores del filtrado adaptativo, no debemos perder de la cabeza el verdadero significado de esta maravillosa herramienta. En la actualidad el filtrado adaptativo tiene múltiples usos, entre los que se encuentran por ejemplo:

- Cancelación de interferencias: Se emplea para cancelar una señal de interferencia desconocida que perturba la información contenida en la señal principal.
- Identificación de sistemas: Se utiliza para los múltiples casos en los que no se dispone de un modelo matemático que caracterice a un sistema. Al usar un filtro adaptativo obtenemos un modelo que permite una buena caracterización del sistema bajo estudio.
- Predicción: Lo usamos también para poder predecir valores futuros de una señal deseada a partir de valores anteriores.
- Igualación: En este caso el objetivo del filtro es aportar un sistema inverso a uno dado del que no se dispone información estadística para su caracterización. Este es el ejemplo que se persigue en el presente proyecto.

Ya sabemos la importancia que tienen las aplicaciones del filtrado adaptativo, no obstante debemos saber elegir que algoritmos adaptativos son más apropiados para cada aplicación en particular. Para hacer una buena elección que suponga un compromiso entre complejidad y resultados aceptables asegurando unas buenas prestaciones, tenemos que atender a las siguientes características [Haykin, 2002]:

- Velocidad de convergencia: Es definido como el número de iteraciones necesarias para que el algoritmo, en un entorno estacionario, converja lo más cerca posible a la solución óptima de Wiener usando el error cuadrático medio como función de coste, partiendo de un filtro inicialmente desajustado.

- Desajuste: Es la desviación que se produce en los algoritmos adaptativos cuando operan en entornos estacionarios, al obtenerse soluciones con error cuadrático medio mayor que el de la solución óptima de Wiener.
- Capacidad de seguimiento: Cuando el filtro adaptativo se encuentra en entornos no estacionarios, es usado para seguir las variaciones estadísticas en el entorno. En este caso el filtro incurre en otro error adicional que tiene como explicación el retardo con el que se alcanza la solución óptima
- Robustez: Se dice de un filtro adaptativo robusto cuando pequeñas perturbaciones (con pequeña energía) suponen únicamente pequeños errores de estimación en la salida.
- Requerimientos computacionales: Se evalúan respecto a varios criterios: el número de operaciones necesarias para completar una iteración del algoritmo; la cantidad de memoria necesaria para almacenar los datos, o la inversión requerida para desarrollar el algoritmo.
- Estructura: Se refiere a que la manera con la que usamos los datos en nuestro algoritmo determina en cierta manera la construcción hardware que se realice.
- Propiedades numéricas: Como propiedades numéricas entendemos dos elementos: ‘Precisión numérica’ aparece como problema en las situaciones en las que la precisión de la representación numérica no es suficiente para obtener resultados fiables cuando se realizan operaciones matemáticas. El otro elemento es la ‘estabilidad numérica’ y es entendida como una característica inherente en el algoritmo adaptativo para que los resultados sean adecuados.
-

# Capítulo 2

## *2.1 Igualación de canal*

En este segundo capítulo vamos a comentar brevemente el problema de igualación de canal, así como las alternativas y las soluciones que se han desarrollado.

El objetivo es obtener un sistema inverso a uno dado, del que no se dispone información estadística necesaria para su caracterización física. Para esta tarea utilizamos los ya comentados filtros adaptativos, el cual modelaremos como el inverso al canal dado, contrarrestando de esta forma su efecto sobre la señal de entrada, a excepción del ruido aditivo el cual no se modela con el filtro adaptativo.

En definitiva, tenemos el problema que dado un canal de comunicaciones con una respuesta al impulso desconocida, el objeto del igualador adaptativo es tratar de asimilar la cascada compuesta por el canal y por el filtro adaptativo para conformar un medio de transmisión ideal [Lucky, 1968; Proakis, 2000].

Para comprenderlo de manera mas clara podemos ver la Figura 2.1, en la que mostramos el esquema de igualación adaptativa, donde $\underline{x}_k = \left[ x_{k-N+1}, x_{k-N+2}, \ \ldots \ , x_k \right]^T$ corresponde con la entrada del filtro, $\underline{h} = \left[ h_{N-1}, h_{N-2}, \ \ldots \ , h_0 \right]^T$ corresponde con los coeficientes del filtro, N la longitud del filtro, $y_k$ la salida del filtro, $e_k$ la salida de error usada para el entrenamiento del filtro, $d_k$ la salida deseada del sistema, y $v_k$ el ruido aditivo gausiano que es

sumado en la cascada. Además tenemos que tener en cuenta que la salida deseada $d_k$ es considerada la entrada retardada en el tiempo.

Como se puede observar en la definición de las variables realizada antes, hemos modificado la variable tiempo del sistema, denominándolo 'k' en vez de 'n' al encontrarnos en tiempo discreto y colocándolo en forma de subíndice en lugar de entre paréntesis. Este cambio es meramente estético y se realiza por facilidad y por el bien de la comprensión de las expresiones, pero el significado continúa siendo el mismo.

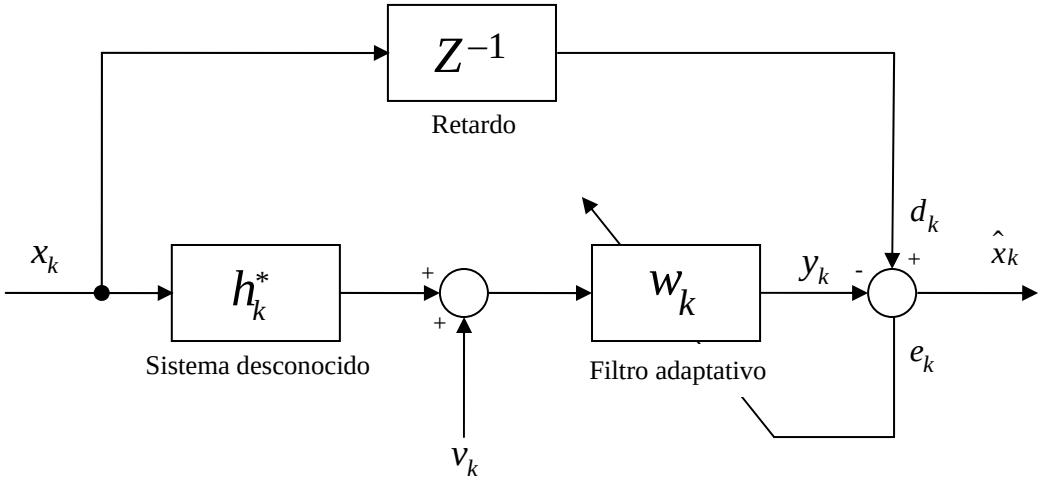



**Figura 2.1 – Esquema de igualación adaptativa de canal**

La salida que obtenemos de nuestro sistema adaptativo para el problema de igualación es la estimación de la entrada al sistema desconocido, $x_k$, y la denotamos como $\hat{x}_k$ (en la Figura 2.1).

Debemos comentar también como modelamos el sistema desconocido en nuestro problema. Tenemos que nuestro sistema se determina con los coeficientes de canal $\underline{h}_k^*$ [1], y mediante $v_k$ que es ruido aditivo gausiano. Tomando estos dos parámetros podemos modelar, usando la suma de convolución característica de los sistemas LTI[2], la expresión 2.1. En definitiva, tenemos que el problema de igualación es aquel que nos permite diseñar una función $\phi(\cdot)$ tal que cumpla la expresión 2.2.

[1] Usamos coeficientes de canal conjugados, ya que contemplamos el caso de valores complejos de entrada y de coeficientes.
[2] Lineales Invariantes en el Tiempo.

$$y_k = \sum_{i=0}^{N-1} h_i^* x_{k-i} + v_k = \underline{h}^H \underline{x}_k + v_k \qquad \textbf{2.1}$$

$$\hat{x}_k = \phi\left(y_{0:k}\right) \longrightarrow y_{0:k} = \{y_0, \ \ldots \ , y_k\} \qquad \textbf{2.2}$$

## *2.2 Igualación óptima*

Conociendo el problema de igualación de canal y su arquitectura, sabemos que podemos resolver el problema de igualación de manera óptima. Suponiendo que la entrada que tenemos es $\underline{x} = \left[x_{-N+1}, \cdots, x_0, \cdots, x_{k+1}\right]^T$ y la salida $\underline{y} = \left[y_0, \cdots, y_{k-1}\right]^T$ podemos determinar que la detección óptima de la entrada $\underline{x}$ a través de su salida $\underline{y}$ corresponde con la expresión que maximiza la probabilidad a posteriori $p\left(x/y\right)$, tal y como se puede ver en la expresión 2.3. A este estimador lo denominamos MAP (máximo a posteriori). Si además, suponemos que la probabilidad de que los símbolos de entrada $\underline{x}$ sean equiprobables, tenemos como resultado que nuestro estimador MAP se convierte en un estimador MV (máxima verosimilitud), que podemos ver en la expresión 2.4.

$$\hat{x}_{MAP} = \arg \max_{\underline{x}} p\left(x/y\right) = \arg \max_{\underline{x}} \left\{ p\left(\underline{y}/\underline{x}\right) p\left(\underline{x}\right) \right\} \qquad \textbf{2.3}$$

$$\hat{x}_{MAP} \xrightarrow{p(\underline{x})=ctte} \arg \max_{\underline{x}} \left\{ p\left(\underline{y}/\underline{x}\right) \right\} = \hat{x}_{MV} \qquad \textbf{2.4}$$

Si usamos notación matricial en la expresión del modelado del canal (Expresión 2.1), podemos llegar a la expresión 2.5, la cual usaremos para reescribir la ecuación del estimador de máxima verosimilitud en la expresión 2.6.

$$y_k = \underline{h}^H \underline{x}_k + v_k \longleftrightarrow \underline{y} = \underline{\underline{H}}^H \underline{x} + \underline{v} \qquad \textbf{2.5}$$

$$\hat{x}_{MV} = \arg \min \left\{ \left\| \underline{y} - \underline{\underline{H}}^H \underline{x} \right\|^2 \right\} \qquad \textbf{2.6}$$

Basándonos en la expresión 2.4, podemos encontrar una solución óptima que satisface la minimización comentada. Esta solución se corresponde con el algoritmo de Viterbi [Viterbi, 1967] que mediante comparación de distancias es capaz de determinar el camino de máxima verosimilitud que detecta a partir de nuestra entrada al filtro, $\underline{y}$ ,nuestra salida estimada, $\underline{x}$, que es la entrada del sistema desconocido.

La mayor ventaja del algoritmo de Viterbi, es que nos ahorra el crecimiento exponencial con ‘k’ (siendo ‘k’ el instante de tiempo de la secuencia) que supondría examinar todos los posibles caminos del diagrama de Trellis de la secuencia de mensajes.

Una contrariedad del algoritmo, es que suponemos que conocemos el canal $\underline{h}^H$ , lo cual es un problema, porque a priori es desconocido según el modelo de igualación de canal. Normalmente se suele obtener mediante la transmisión de unos símbolos piloto, por lo que en la práctica, se supone conocido siempre que el canal no sea demasiado variante.

## *2.3 Estimación de canal*

El desconocimiento del canal puede llegar a ser un problema, ya que introduce un mayor grado de incertidumbre al problema de igualación de canal, y por lo tanto un mayor grado de complejidad. No obstante, en general, se considera conocido, ya que se puede llegar a estimar mediante el uso de unos datos conocidos, denominados patrones de entrenamiento. Al ser conocidos, y sabiendo también el momento en el que son enviados, tenemos tanto la salida del sistema como la entrada, por lo que podemos estimar la respuesta al impulso del canal desconocido, y como consecuencia su comportamiento[3].

A continuación comentaremos las estrategias usadas normalmente para la estimación del canal que usaremos en nuestra arquitectura de igualación.

### 2.3.1 Método de mínimos cuadrados

Tal y como hemos comentado, conocemos tanto la entrada al sistema $\underline{x}_k$, como la salida $y_k$ para el instante de tiempo 'k'. Si además de esto, suponemos que tenemos un canal constante en el tiempo (con esto conseguimos que el modelo que obtenemos sea mas sencillo[4]), tenemos que nuestro estimador corresponderá con la expresión 2.7.

$$\hat{h} = \arg \min_{\underline{h}} \left\{ J\left(\underline{h}\right) = \sum_{k=0}^{K-1} \left| y_k - \underline{h}^H \underline{x}_k \right|^2 \right\} \qquad \textbf{2.7}$$

De la expresión 2.7 podemos comprobar que a más símbolos piloto transmitamos, mejor será nuestra estimación del canal, pero mayor será el malgasto del uso del canal, ya que los símbolos piloto no son información útil. Sin embargo, si no transmitimos los suficientes símbolos piloto, tendremos una situación en la que nuestro canal no será capaz de recuperar bien la información, obteniendo un estimador de canal muy alejado del valor verdadero.

[3] Consideramos que el canal es LTI.
[4] Sin embargo también se puede conseguir un desarrollo del problema para el caso de tener un canal variante en el tiempo, pero será más complejo.

Resolviendo la expresión 2.7 mediante gradientes [Haykin, 2002; Sklar, 1988], obtenemos que la resolución de la expresión 2.7 para el estimador de canal de mínimos cuadrados, es la expresión 2.8.

$$\hat{h} = \left( \sum_{k=0}^{K-1} \underline{x}_k \underline{x}_k^H \right)^{-1} \left( \sum_{k=0}^{K-1} \underline{x}_k \underline{y}_k^* \right) \qquad \textbf{2.8}$$

Podemos determinar a su vez de forma empírica los siguientes elementos de la expresión 2.8:

- Autocorrelación empírica de $\underline{x}_k$ : $\frac{1}{K} \sum_{k=0}^{K-1} \underline{x}_k \underline{x}_k^H = \underline{\underline{\hat{R}}}_{xx}$
- Correlación cruzada empírica de $\underline{x}_k$ e $\underline{y}_k$ : $\frac{1}{K} \sum_{k=0}^{K-1} \underline{x}_k \underline{y}_k^H = \underline{\hat{P}}_{xy}$

### 2.3.1.1 Algoritmo RLS

Una forma de obtener nuestro estimador de la expresión 2.8 que cumple la solución de mínimos cuadrados de forma adaptativa, es el algoritmo Recursive Least Squares (RLS).

El objetivo de este algoritmo es el de aligerar el cálculo del estimador evitando la inversión de la matriz $\underline{\underline{\hat{R}}}_{xx}$, lo cual supondría que la complejidad del estimador crecería con el cubo de las dimensiones de la matriz. Además mediante el uso del RLS, obtenemos la estimación de $\hat{h}$ de manera recursiva.

El desarrollo del algoritmo, se puede encontrar en la bibliografía del proyecto, siendo el resultado iterativo, el que se muestra en las expresiones 2.9 y 2.10.

Inicialización

$$\underline{\hat{h}}(-1) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \textbf{2.9}$$

$$\underline{\underline{\hat{R}}}_{xx}^{-1}(-1) = \gamma \underline{\underline{I}} \quad \gamma = cte$$

Pasos recursivos

$$1- \quad \underline{g}(n) = \frac{\hat{\underline{\underline{R}}}_{xx}^{-1}(n-1)\,\underline{x}_n}{1+\underline{x}_n^H \hat{\underline{\underline{R}}}_{xx}^{-1}(n-1)\,\underline{x}_n}$$

$$2- \quad e_n = \hat{\underline{h}}^H(n-1) - \underline{g}(n) e_n^* \qquad \textbf{2.10}$$

$$3- \quad \underline{\underline{R}}_{xx}^{-1}(n) = \hat{\underline{\underline{R}}}_{xx}^{-1}(n-1) - \underline{g}(n)\,\underline{x}_n^H \hat{\underline{\underline{R}}}_{xx}^{-1}(n-1)$$

La principal ventaja del algoritmo RLS es que converge muy rápidamente, frente a otros algoritmos adaptativos, cuando se utilizan pocas muestras de aprendizaje. Por otro lado, tenemos la desventaja de que es muy sensible a la precisión numérica con la que se trabaja, con lo que podemos incurrir en errores no deseados.

En el caso en el que la hipótesis de entrada sea falsa (hemos supuesto inicialmente que el canal es constante), podemos seguir utilizando el algoritmo RLS, realizando una variación en el algoritmo la cual se traduce en introducir un parámetro de olvido $\lambda$ , tal y como aparece en la expresión 2.11.

$$\hat{\underline{h}}(n) = \arg \ \min \left\{ J(\underline{h}, \lambda) = \sum_{k=0}^{n} \lambda^{n-k} \left| y_k - \underline{h}^H \underline{x}_k \right|^2 \right\} \ \forall \ \ 0 < \lambda < 1 \qquad \textbf{2.11}$$

### 2.3.2 Método de error cuadrático medio

Para obtener la estimación del canal, podemos utilizar otro método además del anteriormente comentado del mínimo error cuadrático. Estamos hablando del método que calcula la función de coste del estimador a partir del error cuadrático medio mínimo. Si presuponemos que tenemos un canal $\underline{h}$ fijo, podemos determinar que la nueva función de coste es la expresión 2.12 la cual satisface el criterio de mínimo error cuadrático medio.

$$\hat{\underline{h}}(n) = \arg \ \min \left\{ J(\underline{h}) = E\left[ \left| y_k - \underline{h}^H \underline{x}_k \right|^2 \right] \right\} \qquad \textbf{2.12}$$

Resolviendo la expresión 2.12, conseguimos obtener la definición de nuestro estimador la cual aparece en la expresión 2.13, siendo el producto de la

inversa de la autocorrelación de los datos de entrada al estimador y la correlación cruzada entre la entrada y la salida.

$$\underline{\hat{h}} = \underline{\underline{R}}_{xx}^{-1} \underline{P}_{xy} \qquad \textbf{2.13}$$

Es importante volver a mencionar que $\underline{\hat{h}}$ no tiene porque coincidir con el propio canal, ya que sólo es una estimación calculada minimizando el error cuadrático medio entre la entrada al sistema $\underline{x}$ y la salida $\underline{y}$, y no estamos recuperando el canal en sentido estricto de la palabra.

### 2.3.2.1 Algoritmo LMS

Al igual que ocurría en el caso del estimador basado en el método de mínimos cuadrados, también tenemos un algoritmo adaptativo para nuestro estimador basado en el error cuadrático medio mínimo. Estamos hablando del algoritmo Least Mean Squares (LMS).

El desarrollo del algoritmo está basado en la aplicación de la función gradiente sobre la expresión 2.13 del estimador, con lo cual se determina que el gradiente de la función de coste $\nabla_{\underline{h}} J$ es aproximadamente igual al gradiente de la función de coste en el instante 'k', $\widehat{\nabla}_{\underline{h}} J_k$. Mediante esta aproximación y utilizando el algoritmo de descenso en la dirección de la máxima pendiente, o gradiente estocástico, obtenemos que la solución iterativa del algoritmo LMS es la que aparece en la expresión 2.14.

$$\begin{gathered} \underline{\hat{h}}(n) = \underline{\hat{h}}(n-1) - \mu \underline{x}_n e_n^* \\ e_n = \underline{\hat{h}}^H (n-1)\underline{x}_n - y_n \\ 0 < \mu << 1 \end{gathered} \qquad \textbf{2.14}$$

Una característica del algoritmo LMS es que cumple el principio de ortogonalidad , o lo que es lo mismo, el algoritmo se para cuando el error que obtenemos en la iteración 'n', $e_n$, es perpendicular a las muestras o con correlación nula, lo que significa que el ruido de desajuste es independiente de la señal de entrada que tengamos.

Como ventaja del algoritmo LMS frente al RLS, podemos decir que el ruido de desajuste es menor si elegimos una $\mu$ suficientemente pequeña. Por el contrario, el algoritmo RLS tiene mayor capacidad de seguimiento que el LMS ya que converge más rápido.

Al igual que ocurría en el caso del algoritmo RLS, también disponemos en LMS de una variante para el caso en el que la hipótesis de partida de que el canal es fijo sea falsa. Bajo estas premisas, obtenemos como resultado la expresión

2.15 que muestra la nueva función de coste, con el parámetro de olvido $\lambda$.

$$\underline{\hat{h}}(n) = \arg \min_{\underline{h}} \left\{ J(\underline{h}, \lambda) = \sum_{k=0}^{n} \lambda^{n-k} \left| y_k - \underline{h}^H \underline{x}_k \right|^2 \right\} \qquad \textbf{2.15}$$
$$0 < \lambda < 1$$

## *2.4 Estimación de canales variantes con el tiempo*

Anteriormente hemos desarrollado expresiones de estimadores de canal los cuales eran principalmente fijos. No obstante hemos dado ciertas variantes para los algoritmos adaptativos RLS y LMS para el caso en que el canal fuera variante en el tiempo. La solución que nos ofrecen estos algoritmos adaptativos no es completa, ya que se basan en una aproximación referenciada al método usado en la definición de la función de coste del problema. En este caso, nos proponemos calcular la solución exacta a la hipótesis inicial de que el canal es variante. Esta solución se denomina filtro de Kalman a nombre de su autor.

El filtro de Kalman, se define mediante una ecuación de estado, expresión 2.16, y una ecuación de observación, expresión 2.17. Tal y como se puede determinar con el nombre de las ecuaciones, la ecuación de estado nos permite determinar el estado de la entrada de forma iterativa, sumando un determinado ruido $\underline{v}_t$ que corresponde con una variable estadística normal de media nula y matriz de covarianza $\underline{\underline{Q}}_t$ en el instante 't'. Por otro lado, tenemos la ecuación de observación que determina la salida de nuestro sistema, dependiente nuestro estado instantáneo $\underline{x}_t$. Además, tenemos a su vez otro término de error, $\underline{w}_t$, que es al igual que $\underline{v}_t$, una distribución normal de media nula y con matriz de covarianza $\underline{\underline{R}}_t$.

En estas ecuaciones debemos destacar que son lineales y gaussianas, y que de partida $\underline{\underline{F}}$, $\underline{\underline{Q}}$ y $\underline{\underline{R}}$ son conocidas. La observación la obtenemos multiplicando el estado $\underline{x}_t$ desconocido, multiplicándolo por la matriz conocida $\underline{\underline{H}}$. Tomamos como valor inicial la hipótesis de que el estado en reposo es $\underline{x}_{-1} \sim N\left(\underline{x}_{-1}; \hat{\underline{x}}_{-1}, \underline{\underline{P}}_{-1}\right)$.

$$\underline{x}_t = \underline{\underline{F}}_{t-1}\underline{x}_{t-1} + \underline{v}_{t-1} \text{ con } \underline{v}_t \sim N\left(0, \underline{\underline{Q}}_t\right) \qquad \textbf{2.16}$$

$$\underline{z}_t = \underline{\underline{H}}_{t-1}\underline{x}_t + \underline{w}_t \text{ con } \underline{w}_t \sim N\left(0, \underline{\underline{R}}_t\right) \qquad \textbf{2.17}$$

Desarrollando la densidad de probabilidad $P\left(\underline{x}_{t-1}/\underline{z}_{0:t-1}\right)$ de manera exacta y recursivamente, podemos obtener que la predicción corresponde con la que aparece en la expresión 2.18:

$$\begin{aligned}
&\textit{predicción}\left\{\begin{array}{l}\hat{\underline{x}}_{t/t-1}=\underline{\underline{F}}_{t-1}\hat{\underline{x}}_{t-1/t-1}\\ \underline{\underline{P}}_{t/t-1}=\underline{\underline{Q}}_{t-1}+\underline{\underline{F}}_{t-1}\underline{\underline{P}}_{t-1/t-1}\underline{\underline{F}}_{t-1}^{H}\end{array}\right\}\\
&\textit{actualización}\left\{\begin{array}{l}\hat{\underline{x}}_{t/t}=\hat{\underline{x}}_{t/t-1}+\underline{\underline{K}}_{t}\left(\underline{z}_{t}-\underline{\underline{H}}_{t}\hat{\underline{x}}_{t/t-1}\right)\\ \underline{\underline{P}}_{t/t}=\underline{\underline{P}}_{t-1/t}-\underline{\underline{K}}_{t}\left(\underline{\underline{H}}_{t}\underline{\underline{P}}_{t-1/t}\underline{\underline{H}}_{t}^{H}+\underline{\underline{R}}_{t}\right)\underline{\underline{K}}_{t}^{H}\end{array}\right\}\\
&\textit{Ganancia de Kalman}\left\{\underline{\underline{K}}_{t}=\underline{\underline{P}}_{t/t-1}\underline{\underline{H}}_{t}^{H}\left[\underline{\underline{H}}_{t}\underline{\underline{P}}_{t/t-1}\underline{\underline{H}}_{t}^{H}+\underline{\underline{R}}_{t}\right]^{-1}\right\}
\end{aligned} \qquad \textbf{2.18}$$

Si aplicamos esta solución a un modelo de variación de canal, del tipo de la expresión 2.19, modelado como un proceso AR de orden $L\geq 1$ y con un ruido blanco[5] $\underline{v}_k\sim N\left(0,\sigma_v^2\underline{\underline{I}}_L\right)$, tenemos que usando la matriz $\underline{\underline{D}}$ (multiplicador de los coeficientes de canal) y el vector $\underline{\tilde{h}}_k$ (coeficientes de canal) definidos en 2.20, conseguimos obtener las ecuaciones de Kalman de estado y observación mostradas en la expresión
2.21.

$$\underline{h}_k=\sum_{l=1}^{L}\gamma_l\underline{h}_{k-1}+\underline{v}_k \qquad \textbf{2.19}$$

$$\underline{\tilde{h}}_k=\begin{bmatrix}\underline{h}_{k-L+1}\\ \vdots\\ \underline{h}_k\end{bmatrix}_{NL\times 1}\qquad \underline{\underline{D}}=\begin{bmatrix}\gamma_L\underline{\underline{I}}_n & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \gamma_1\underline{\underline{I}}_n\end{bmatrix} \qquad \textbf{2.20}$$

$$\begin{aligned}
&\text{Ec. estado}\ \left\{\underline{\tilde{h}}_k=\underline{\underline{D}}\underline{\tilde{h}}_{k-1}+\underline{\tilde{v}}_k\right\}\\
&\text{Ec. observación}\ \left\{z_k=\underline{x}_k^H\underline{\underline{c}}\underline{\tilde{h}}_k+u_k\right\}
\end{aligned} \qquad \textbf{2.21}$$

Con estas ecuaciones de la solución de Kalman aplicadas al modelo de estimación de canal variante, podemos rehacer las expresiones anteriormente comentadas en 2.18, y que aparecen en 2.22.

[5] $\underline{\underline{I}}_L$ corresponde con la matriz identidad de dimensión 'L'.

$$
\begin{aligned}
&\textit{Inicialización}\ \left\{\begin{matrix}\underline{\underline{P}}_{-1}=\sigma^2\underline{\underline{I}}\\ \underline{\hat{h}}_{-1}=\varnothing\end{matrix}\right\}\underline{\hat{h}}_{-1}\sim N\left(0,\sigma^2\underline{\underline{I}}\right)\\
&\textit{predicción}\left\{\begin{matrix}\underline{\hat{h}}_{k/k-1}=\underline{\underline{D}}\,\underline{\hat{h}}_{k-1/k-1}\\ \underline{\underline{P}}_{k/k-1}=\underline{\underline{D}}\,\underline{\underline{P}}_{k-1/k-1}\underline{\underline{D}}^{H}+\sigma_v^2\underline{\underline{I}}\end{matrix}\right\}\\
&\textit{actualización}\left\{\begin{matrix}\underline{\hat{h}}_{k/k}=\underline{\hat{h}}_{k/k-1}+\underline{\underline{K}}_k\left(z_k-\underline{x}_k^{H}\underline{\underline{C}}\,\underline{\hat{h}}_{k/k-1}\right)\\ \underline{\underline{P}}_{k/k}=\left(\underline{\underline{I}}-\underline{\underline{K}}_k\underline{x}_k^{H}\underline{\underline{C}}\right)\underline{\underline{P}}_{k-1/k}\end{matrix}\right\}\\
&\textit{Ganancia de Kalman}\left\{\underline{\underline{K}}_k=\underline{\underline{P}}_{k/k-1}\underline{\underline{C}}^{H}\underline{x}_k\left(\underline{x}_k^{H}\underline{\underline{C}}\,\underline{\underline{P}}_{k/k-1}\underline{\underline{C}}^{H}\underline{x}_k+\sigma_u^2\right)^{-1}\right\}
\end{aligned}
\qquad \textbf{2.22}
$$

Con esta solución, si tenemos una secuencia de entrenamiento $\underline{x}_k$, o en su defecto los $\underline{x}_k$ se reciben muy bien, podemos obtener un vector de estimación de canal óptimo.

## *2.5 Estimación conjunta del canal y los datos*

En este tipo particular de estimación, nuestro objetivo es obtener una estimación tanto de la secuencia de entrada de nuestro sistema, como del canal a través del cual estamos haciendo pasar la información. De esta manera podemos decir que tanto los datos de entrada $\underline{x}_k$, como el vector de canal $\underline{h}$ son desconocidos a priori en el planteamiento de nuestro problema, por esta razón, a este tipo de algoritmo de igualación se le suele denominar, como algoritmo 'ciego'.

Como tal, denominaremos algoritmo 'ciego' a aquel que no necesita de secuencia de entrenamiento. Los principios de esta metodología de problema se pueden obtener a través de las referencias bibliográficas [Seshadr, 1994 ; Raheli, Polydros, Tzon, 1995].

El funcionamiento principal del algoritmo consiste en que según se determina la secuencia de transmisión, se le asocia una estimación de canal asociada. Para aquellas posiciones de bit donde no haya error, la estimación de canal será de mínimos cuadrados.

El algoritmo que estamos tratando se le denomina PSP en siglas de "Per Survivor Processing" o "Procesado por Supervivientes". Es semejante al algoritmo de Viterbi anteriormente comentado, con la única salvedad de que en este caso, en vez de tener un solo superviviente para cada estado, guardamos unos pocos y estimamos el canal por cada tira de bits obteniendo una estimación de canal, con un coste intuitivo. El camino correcto tendrá la secuencia y estimación de canal con el coste mas bajo. En principio, para cada iteración en cada estado guardamos n supervivientes (el motivo de limitar los supervivientes por iteración es debido a que la complejidad se incrementa de manera exponencial con el número de supervivientes elegidos) y con cada uno, tenemos una estimación de canal que se actualiza en cada iteración (para al actualización de la estimación de canal, podemos utilizar cualquiera de los algoritmos adaptativos comentados en los puntos anteriores, como por ejemplo el RLS).

Entrando más en detalle en el algoritmo, tenemos que la base es un conjunto de estados similares a los empleados en el algoritmo de Viterbi,

denominados $e_k^{(i)}$, donde 'i' determina el número de estado y 'k' el instante de tiempo al cual pertenece. Por otro lado tenemos la secuencia de símbolos asociada a los estados $e_k^{(i)}$, que es $x_{0:k}^{(i)}$, la secuencia de observaciones $y_{0:k}$, y la estimación de canal obtenida en cada instante 'k' $\hat{\underline{h}}_k^{(i)} = \phi\left(y_{0:k}, x_{0:k}^{(i)}\right)$. Hemos de tener en cuenta que tenemos una estimación de canal para cada estado 'i', ya que para cada posible combinación de estados calculamos la estimación resultante.

Como variables auxiliares que nos permitirán obtener nuestro resultado, se define $\lambda\left[e_k^{(i)} \to e_{k+1}^{(j)}\right]$ como el coste de pasar del estado i-esimo en el instante 'k' al estado j-esimo en el instante 'k+1'. A su vez se define también $\Gamma\left[e_k^{(i)}\right]$ como el coste acumulado hasta el estado $e_k^{(i)}$.

Una vez planteadas todas las variables de nuestro sistema podemos desarrollar los procedimientos del algoritmo, que en fundamento es la base de Viterbi sobre el conjunto de estados representados en un diagrama Trellis. Los pasos del algoritmo se detallan en 2.23.

$$\begin{aligned} &1 \;\to\; \text{Calcular } \lambda\left[e_k^{(i)} \to e_{k+1}^{(j)}\right] \text{ para cada posible transición} \\ &2 \;\to\; \text{Determinar los supervivientes} \begin{cases} S_k^{(i)} = \arg\min_{e_k}\left\{\Gamma\left[e_k\right] + \lambda\left[e_k \to e_{k+1}\right]\right\} \\ \Gamma\left[e_{k+1}^{(i)}\right] = \Gamma\left[S_k^{(i)}\right] + \lambda\left[S_k^{(i)} \to e_{k+1}^{(i)}\right] \end{cases} \end{aligned} \qquad \textbf{2.23}$$

Si expandimos el desarrollo del algoritmo mostrado en la expresión 2.23, centrándonos en el algoritmo PSP, podemos añadir los pasos mostrados en 2.24 como continuación de los de 2.23.

$$\begin{aligned} &3 \;\to\; \text{Actualización del canal para cada superviviente} \\ &\qquad \hat{\underline{h}}_{k+1}^{(i)} = \hat{\underline{h}}_k^{(\cdot)} - \mu \underline{x}_{k+1}^{(i)} \left[\hat{\underline{h}}_k^{(\cdot)^H} \underline{x}_{k+1}^{(i)} - y_{k+1}\right]^* \\ &4 \;\to\; \text{Costes acumulados} \end{aligned} \qquad \textbf{2.24}$$

$$\Gamma\left[e_{k+1}^{(i)}\right] = \Gamma\left[S_k^{(i)}\right] + \left| y_{k+1} - \underline{\hat{h}}_{k+1}^{(i)^H} \underline{x}_{k+1}^{(i)} \right|^2$$

Como podemos ver en 2.24, el punto 3 se corresponde con la actualización de la estimación de canal basada en un algoritmo adaptativo (en este caso un LMS)[6].

Una contrariedad de este tipo de algoritmos es que cuando la ISI[7] es muy severa escalan mal y nos obliga a aumentar mucho el número de supervivientes por estado. Esta situación introduce una complicación que nos limita muy directamente para llegar al Viterbi óptimo bajo las circunstancias de canal conocido.

[6] Usamos en la lista de pasos el algoritmo LMS que se basa únicamente en la última estimación de canal, pero podríamos usar el RLS guardando además la ultima inversa de la matriz de autocorrelación, o el filtro de Kalman guardando además la media y la varianza de la matriz de autocorrelación.

[7] Interferencia Inter Simbolica.

## *2.6 Igualadores lineales y realimentados*

Este tipo de igualadores se utilizan en lugar del Viterbi con el objetivo de reducir la complejidad del citado algoritmo. En base, utilizan las observaciones del sistema de forma lineal y de los símbolos detectados de forma realimentada. Tal y como comentamos en la introducción podemos tener dos tipos de igualadores, dependiendo de las muestras que usemos para decidir, hasta el instante 'k' sin retardar, o hasta el instante 'k+d' introduciendo un retardo de decisión, tal y como se muestra en la Figura 2.2.

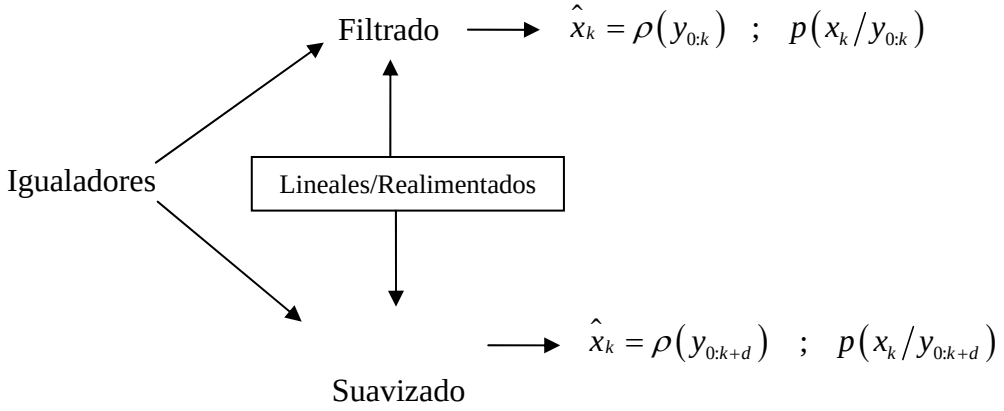



**Figura 2.2 – Igualadores lineales y realimentados**

En cualquier caso, ambas técnicas contempladas en la Figura 2.2 son intercambiables, es decir, un filtrado se puede extender a un suavizado, y por su lado, un suavizado se puede restringir a un filtrado.

### 2.6.1 Filtrado con ISI nula

En este punto vamos revisar las alternativas existentes que se han desarrollado para evitar la distorsión producida por la ISI en los procesos de filtrado.

#### 2.6.1.1 Deconvolución lineal del canal

En este supuesto, nos encontramos bajo las condiciones iniciales de que no tenemos ruido en nuestro sistema, y nuestro objetivo es el de realizar una suma de convolución en tiempo discreto. A partir de esta premisa intentamos

llegar a la expresión de nuestro estimador lineal que aparece en la expresión 2.25 que no es más que una convolución lineal de las observaciones.

$$y_k = \sum_{i=0}^{N-1} h_i^* x_{k-i} \;\rightarrow\; \hat{x}_k = \frac{1}{h_0^*}\left( \sum_{i=1}^{N-1} h_i^* \hat{x}_{k-i} - y_k \right) \qquad \textbf{2.25}$$

Implementando nuestro sistema nos encontramos que es básicamente un filtro FIR, tal y como aparece en la Figura 2.3.

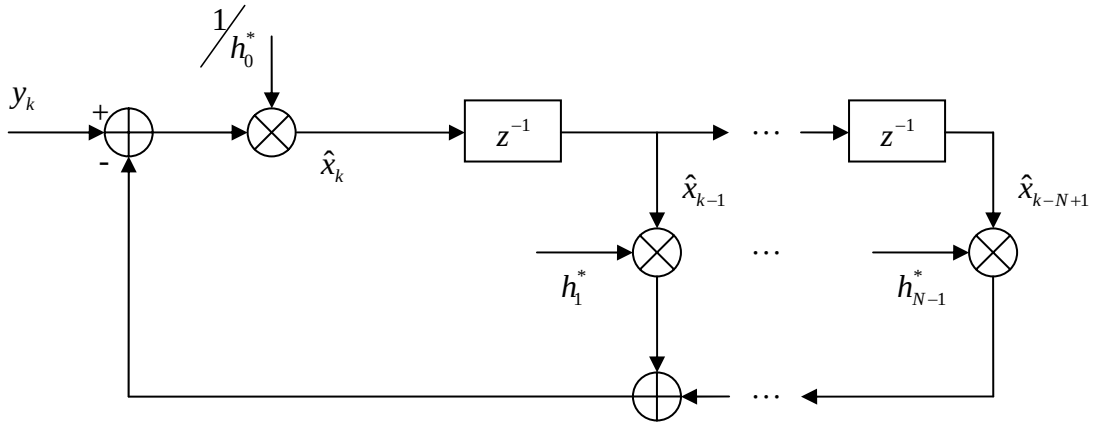



**Figura 2.3 – Sistema de deconvolución lineal.**

Desarrollando la expresión 2.25, tenemos que el montaje de la Figura 2.3 cancela totalmente la interferencia intersimbólica, tanto para las suposiciones iniciales de existencia o no, de ruido a la entrada. En el caso de ruido en particular, encontramos que la estimación de salida de nuestro montaje, $\hat{x}_k$, viene compuesta por el símbolo transmitido, $x_k$, y por un termino correspondiente de error, $\tilde{u}_k$ que vendrá determinado por el ruido gaussiano que tenemos a la entrada de nuestro sistema.

### 2.6.1.2 Decision Feedback Equalizer (DFE) con ISI nula.

En este nuevo tipo de igualador tenemos una importante novedad, que se fundamenta principalmente en que las decisiones del símbolo transmitido será la información que se realimente en la implementación del igualador, tal y como se muestra en la Figura 2.4.

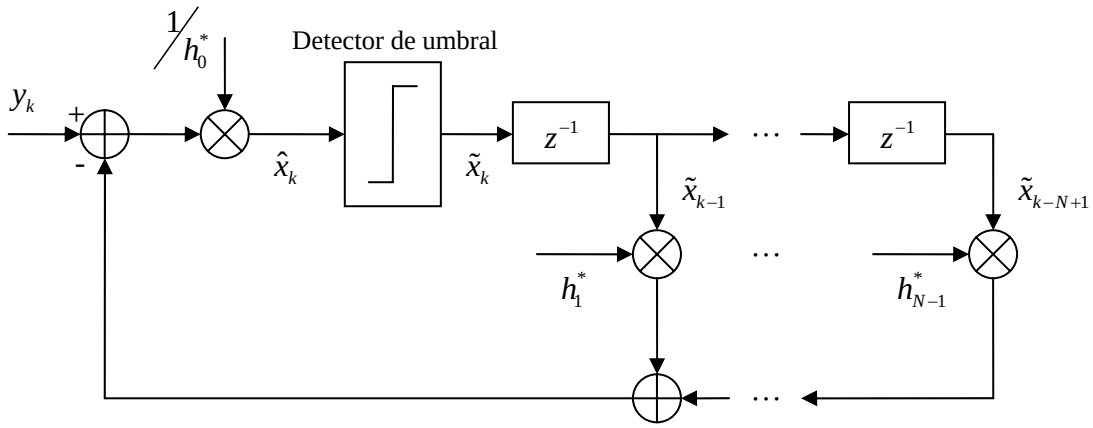



**Figura 2.4 – Igualador de realimentación con ISI nula**

La única diferencia con el igualador anterior, comentado en el apartado 2.6.1.1, es la pérdida de linealidad en el montaje, ya que en este caso la realimentación se realiza sobre los símbolos estimados, $\tilde{x}_k$, y no sobre los símbolos transmitidos, $\hat{x}_k$, como ocurría en el caso anterior. Por esta razón, introducimos en el esquema de la Figura 2.4 el elemento detector de umbral que nos permite estimar los símbolos frente a los símbolos que transmitimos. En este caso, tenemos una estimación dura, debido al detector de umbral, en contra de la estimación blanda que teníamos en el igualador lineal.

Con el fin de poder caracterizar el funcionamiento de nuestro sistema, tenemos que cuando la probabilidad de error es mucho menor que 0.5, o lo que es lo mismo, $P_e << \frac{1}{2}$, el detector de umbral es capaz de suprimir la ISI, suponiendo que tenemos una buena estimación de canal. El problema nos aparece cuando la $P_e$ no es tan baja, ya que en lugar de quitar interferencia, la añadimos y $\hat{x}_k$ puede ser de mala calidad. Este último apunte, nos da lugar a un problema de propagación de errores, ya que ensuciamos nuestras decisiones en función de las malas decisiones que tomamos en el pasado, y de esta forma aumentamos la $P_e$.

En general, esta situación comentada aparece en el caso de que tengamos un $h_0$ no muy grande, ya que eso implica que no tenemos línea directa en el canal, y por lo general tendremos malas estimaciones, porque nos faltará información. En condiciones normales con línea de visibilidad directa casi funciona en los mismos márgenes que el Viterbi con mucha menor complejidad.

### 2.6.2 Técnicas de suavizado

En este punto introduciremos las alternativas que se han desarrollado en suavizado. Tal y como podemos ver en la Figura 2.2, la principal diferencia con respecto al proceso de filtrado es que en este caso se introduce un retardo que permite disponer de mayor información para nuestras estimaciones.

#### 2.6.2.1 Suavizador MMSE lineal

En este caso tenemos una metodología semejante a la comentada en el caso de filtrado lineal basado en deconvolución, ya que como hemos comentado, tenemos una estimación blanda en la realimentación del sistema.

En el desarrollo de las expresiones del suavizador suponemos una entrada $\underline{y}_k$ al sistema con unos pesos $\underline{w}$, y empleando para la realización de la función de coste el criterio de error cuadrático medio, tenemos que el resultado es lo que mostramos en la expresión 2.26.

$$\underline{y}_k = \left[y_{k-a}, \cdots, y_k, \cdots, y_{k+a}\right]^T \quad \underline{w} = \left[w_{2a}, \cdots, w_0\right]^T$$
$$\hat{x}_k = \underline{w}^H \underline{y}_k \qquad \textbf{2.26}$$
$$\hat{\underline{w}} = \arg\min_{\underline{w}} \left\{ J\left(\underline{w}\right) = E\left[\left| x_k - \hat{x}_k \right|^2\right] \right\}$$

Como podemos ver en la función de coste de $\hat{\underline{w}}$, tenemos que la expresión es cuadrática, con lo que tenemos una función convexa y por tanto con una única solución.

Desarrollando la expresión 2.26, tenemos que la solución de nuestro suavizador corresponde con la solución de Wiener varias veces mencionada en el presente proyecto, y que podemos ver en la expresión 2.27.

$$\hat{\underline{w}} = \left(E\left[\underline{y}_k \underline{y}_k\right]\right)^{-1} E\left[\underline{y}_k x_k^*\right] \qquad \textbf{2.27}$$

Si estimamos la autocorrelación, así como la correlación cruzada entra la entrada y la salida, podemos reescribir la expresión de nuestro suavizador para obtener otra más efectiva y realizable, y que se muestra en la expresión 2.28.

$$\hat{\underline{w}} = \sigma_x^2 \left( \sigma_x^2 \underline{\underline{H}}_a^H \underline{\underline{H}}_a + \sigma_u^2 \underline{\underline{I}} \right)^{-1} \underline{\underline{H}}_a^H \alpha_{a+N} \qquad \textbf{2.28}$$

En la expresión 2.28 podemos encontrar que tenemos la varianza $\sigma_x^2$, que corresponde con la varianza de los símbolos de entrada, $\sigma_u^2$ que es la varianza del ruido, $\underline{\underline{H}}_a$ que es una matriz que corresponde con la variación de canal para cada uno de los '2a+1' entradas que tenemos en nuestro suavizador, y $\alpha_{a+N}$ que se corresponde con un vector de ceros con un uno en la posición 'a+N'.

### 2.6.2.2 Suavizador DFE-MMSE

Al igual que ocurría en el caso del filtrado, el esquema DFE nos permitía calcular nuestro estimador utilizando la realimentación en nuestro sistema, y basándonos de nuevo en este procedimiento construimos nuestro suavizador, cuya expresión aparece en 2.29.

$$\hat{x}_k = \underline{w}^H \underline{y}_k - \underline{v}^H \underline{\tilde{x}}_k \qquad \textbf{2.29}$$

Tal y como podemos observar en 2.29, tenemos dos direcciones en nuestra ecuación, una hacia delante cuyos pesos son $\underline{w}$ que se alimentan de las entradas a nuestro sistema, y otra dirección hacia detrás, cuyos pesos son $\underline{v}$ y que se alimentan, en este caso, de los datos estimados en nuestro sistema.

En este caso nuestra función de coste vendrá determinada por estos nuevos parámetros de diseño que hemos introducido en nuestro nuevo suavizador, tal y como podemos ver en la expresión 2.30.

$$\left( \hat{\underline{w}}, \hat{\underline{v}} \right) = \arg \min_{\left( \hat{\underline{w}}, \hat{\underline{v}} \right)} \left\{ J\left( \underline{w}, \underline{v} \right) = E\left[ \left| x_k - \hat{x}_k \right|^2 \right] \right\} \qquad \textbf{2.30}$$

La Figura 2.5 nos muestra la implementación del suavizador propuesto, en el que aparece que la realimentación con pesos $\underline{v}$ es la encargada de tomar la interferencia causal del sistema y realimentarla para combinándola, eliminar su contribución en la dirección hacia delante del esquema (la rama del esquema relacionada con la entrada del sistema, $\underline{y}_k$).

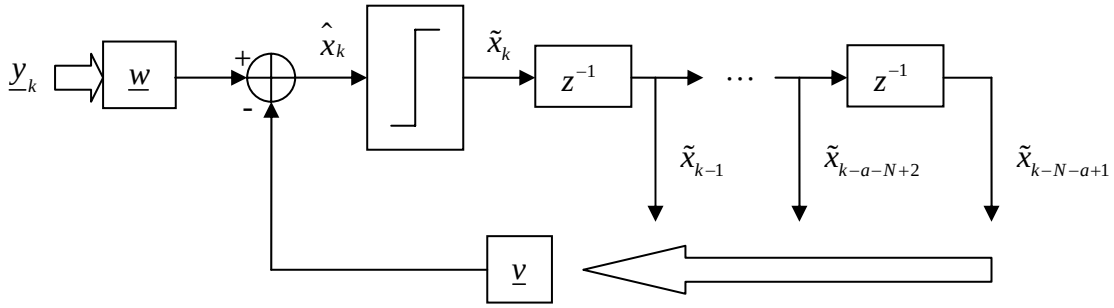



**Figura 2.5 – Esquema de suavizador DFE-MMSE.**

El problema en este tipo de suavizadores es que los pesos $\underline{w}$ y $\underline{v}$, se encuentran relacionados directamente, con lo cual es más complicado el ajuste del sistema completo.

## *2.7 Turbo igualación*

El esquema de turbo igualación persigue la finalidad de detectar obteniendo las probabilidades de los datos que estamos transmitiendo, hasta que las probabilidades a posteriori se mantienen constantes. Para realizar este seguimiento de las probabilidades a posteriori de los datos enviados por nuestro sistema, disponemos de un algoritmo MAP[8], denominado BJCR en honor de sus descubridores y que detallaremos en los siguientes puntos. No obstante antes de introducirnos en el algoritmo y su funcionamiento debemos comentar el esquema de turbo igualación, que podemos ver en la Figura 2.6.

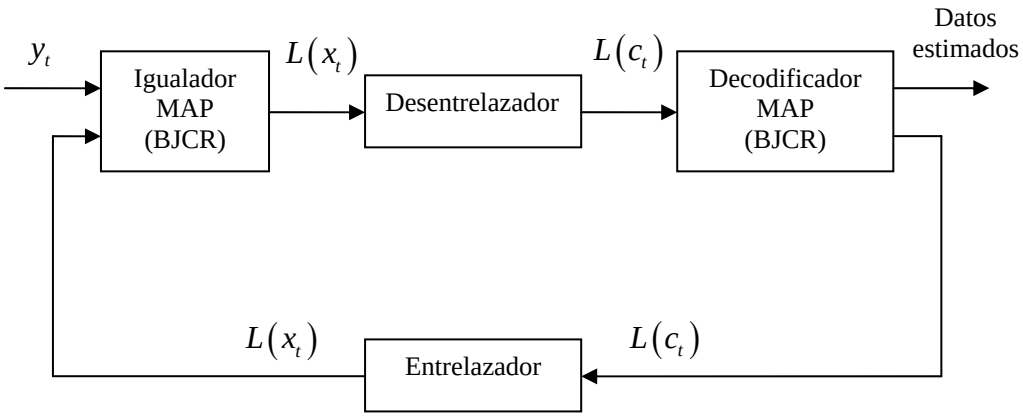



**Figura 2.6 – Esquema de turbo igualación.**

En el esquema mostrado en la Figura 2.6, debemos mencionar que el igualador MAP es equivalente a la introducción de un código convolucional. La salida de este código será entrelazada con la finalidad de dispersar los posibles errores y que no recaigan sobre grupos de símbolos consecutivos. Posteriormente se decodifica nuestro código convolucional, obteniendo de esta manera los datos estimados mediante el decodificador MAP.

Otro apunte importante del esquema mencionado es su funcionamiento ya que es símbolo a símbolo y de manera iterativa. Para ello se modela el canal como un proceso Markoviano [Proakis, 2001] en el que tenemos una entrada $x_t$ y una salida $y_t$, y el cual viene gobernado por un conjunto de estados $\left[x_{t-N+1},\cdots,x_{t-1}\right]^T$ que nos dicen en que situación se encuentra nuestro canal para

[8] Máximo a posteriori.

cada instante 't'. A partir de este diagrama de estados se construye un diagrama Trellis y se realiza la codificación/decodificación símbolo a símbolo.

### 2.7.1 Algoritmo BJCR

Este algoritmo recibe el nombre de sus creadores Bahl-Cocke-Jelinek-Raviv, y su funcionamiento a grandes rasgos es como un Viterbi, pero realizando dos pasadas, una hacia delante y otra hacia atrás.

Se apoya sobre la suposición de que nuestra fuente de datos es markoviana modelada con M posibles estados y nuestro canal DMC[9] sin memoria con lo que la probabilidad de una trama de símbolos es el producto de las probabilidades individuales. De esta forma tenemos el esquema que se muestra en la Figura 2.7.

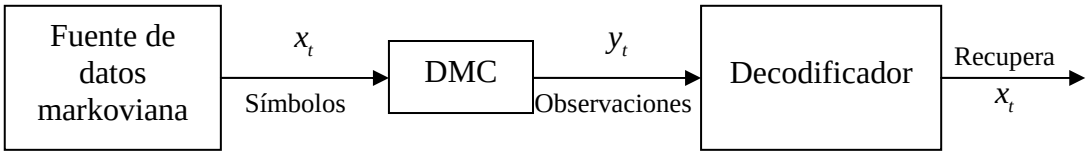



**Figura 2.7 – Esquema del algoritmo BJCR.**

Bajo estas premisas, se desarrolla el algoritmo utilizando una notación específica para determinar las variaciones de estado de la fuente markoviana. Esta notación la definimos en la expresión 2.31.

$$\begin{aligned} &\text{Probabilidad de estar en el estado m: } \lambda_t(m) = \Pr(S_t = m; y_{1:t}) \\ &\text{Probabilidad de cambiar del estado m' al m: } \sigma_T(m', m) = \Pr(S_t = m; S_{t-1} = m'; y_{1:T}) \end{aligned} \qquad \textbf{2.31}$$

Además, tenemos que estas variables usadas en algoritmo y definidas en la expresión 2.31 se apoyan sobre otras variables auxiliares que detallamos en la expresión 2.32.

[9] Se refiere a un canal discreto sin memoria. Su funcionamiento es que cuando tiene un símbolo a la entrada, a la salida da otro símbolo con una determinada probabilidad.

Probabilidad de estar en el estado m utilizada en la pasada hacia delante del Viterbi:

$$\alpha_t(m) = \Pr(S_t = m; y_{1:t})$$

Probabilidad de símbolo condicionada a encontrarnos en el estado m. Es utilizado en la pasada hacia atrás:

$$\beta_t(m) = \Pr(y_{t+1:T} / S_t = m) \qquad \textbf{2.32}$$

Probabilidad de cambiar del estado m' al m símbolo a símbolo:

$$\gamma_t(m', m) = \Pr(S_t = m; y_t / S_{t-1} = m')$$

Además de estas variables auxiliares, tenemos las relaciones entre las variables principales del algoritmo, mostradas en 2.31 y las auxiliares mostradas en 2.32. Dichas relaciones las mostramos en la expresión 2.33.

$$\begin{aligned}
\lambda_t(m) &= \beta_t(m)\alpha_t(m) \\
\sigma_T(m', m) &= \beta(m)\gamma_t(m', m)\alpha_{t-1}(m') \\
\alpha_t(m) &= \sum_{m'} \gamma_t(m', m)\alpha_{t-1}(m') \\
\beta_t(m) &= \sum_{m'} \beta_{t+1}(m')\gamma_{t+1}(m', m) \\
\gamma_t(m', m) &= P_t(m/m')\Pr(y_t / S_t = m, S_{t-1} = m')
\end{aligned} \qquad \textbf{2.33}$$

Una vez comentada todas las variables que intervienen en el algoritmo, estamos en disposición de detallar los pasos:

- Inicialización: Se supone un estado inicial 0, en el que $\alpha_0(0) = 1$ , $\alpha_0(m > 0) = 0$ y $\beta_T(0) = 1, \beta_T(m > 0) = 0$.
- Realizamos una pasada hacia delante: Calculamos $\alpha_t(m)$ y $\gamma_t(m', m)$ con las expresiones de 2.33.
- Una vez recibido $y_{1:T}$ , hemos llegado al final de nuestras observaciones y hacemos la pasada hacia atrás: Calculamos $\beta_t(m)$ con la expresión de 2.33.
- Finalizamos: Una vez terminada la pasada hacia atrás, calculamos $\lambda_t(m)$ y $\sigma_T(m', m)$ (Probabilidad a posteriori de los símbolos) con las expresiones de 2.33.

Como ultimo apunte del algoritmo, debemos decir que el BCJR tiende a la detección óptima de secuencia que se realiza con el Viterbi, siendo en nuestro

caso el detector óptimo de cada símbolo, al obtener las probabilidades de cada símbolo de la secuencia. El contrapunto del método, es que tenemos complejidad doble con respecto al Viterbi, lo que no suele ser ventajoso, porque nos interesa principalmente tener un detector MAP de la trama y no de los símbolos sueltos.

# Capítulo 3 Estado del arte

## *3.1 Introducción*

En este tercer capítulo vamos a hacer un recorrido sobre la actualidad del problema de igualación de canal, haciendo especial hincapié en el caso no lineal, no comentado en el capítulo anterior con demasiada profundidad, y que protagoniza el motivo del presente proyecto fin de carrera.

Organizaremos el capítulo en dos grandes grupos, el primero de ellos ocupado de mostrar brevemente el estado actual de los algoritmos mostrados en el capitulo 2 sobre igualación lineal, así como sus prestaciones. En el segundo bloque realizaremos un muestrario de los diferentes métodos empleados en la actualidad para resolver el problema de igualación de manera no lineal, y los fundamentaremos con ejemplos prácticos que refuercen nuestras explicaciones.

## *3.2 Situación de los igualadores lineales*

En la actualidad los igualadores son utilizados en multitud de aplicaciones, no obstante, no es nuestro objetivo el comentar todas las prestaciones de esos igualadores en cada una de las aplicaciones comerciales del mercado. Nuestro objetivo en este caso es tomar una aplicación concreta, lo suficientemente representativa de la importancia del uso de un buen igualador y con un canal lo mas variable posible, para tener de esa forma un caso complicado de igualación que en cierta medida pueda englobar el funcionamiento de todas las demás aplicaciones.

El entorno que hemos elegido para mostrar la penetración que tienen los igualadores comentados en el capítulo anterior, son los sistemas comerciales de telefonía móvil, ya que en la mayor parte de ellos, se emplean técnicas de igualación de canal para garantizar la calidad de la transmisión, en términos de SER[1].

Otro motivo que nos favorece la elección de este escenario, es la pluralidad de métodos que existen, ya que el procedimiento de igualación generalmente no se encuentra especificado por ninguna norma, y se deja a la libre elección del fabricante de los equipos.

Vamos a ir comentando, cada tecnología móvil desplegada, detallando cual es el mecanismo de igualación empleado, en caso de ser especificado mediante norma (tal y como hemos comentado, si no está por norma depende de cada fabricante en particular), así como la estimación de canal que se realiza.

- GSM: El igualador no se encuentra especificado. Con respecto a la estimación de canal, tenemos que se dispone de 26 bits de entrenamiento que se encuentran situados en el centro de la trama normal, y que son usados para la estimación de canal. Comúnmente se han utilizado igualadores de Viterbi y DFE.

[1] Se refiere al término del inglés “Symbol Error Rate”.

- UMTS: Posee una norma muy abierta que contempla tanto la igualación de canal a partir de símbolos piloto/secuencias de entrenamiento, como la cancelación de interferencias de acceso múltiple. Generalmente se combinan ambos aspectos usando métodos de tipo MMSE o mínimos cuadrados.

- IS-54, IS-136: Son sistemas de telefonía móvil de segunda generación, conocidas como Digital-AMPS y que se han desplegado en EEUU y Canada principalmente. Utilizan tecnología TDMA en el acceso al medio. Al igual que ocurría en el caso de GSM, el igualador es propietario y no se encuentra especificado. No obstante en la red USDC[2] se utilizan igualadores DFE con coeficientes calculados mediante un algoritmo RLS.

- IS-95: Estándar de telefonía móvil de segunda generación basado en tecnología CDMA y cuyo nombre comercial es cdmaOne. Ha sido reemplazado por CDMA2000 que coexiste con UMTS. Su diseño requiere estimación de canal de mínimos cuadrados a partir de una secuencia de entrenamiento, ya que utiliza un receptor RAKE de tres ramas con combinación coherente.

- PDC: Tecnología móvil de segunda generación que se utiliza en comunicaciones móviles digitales en Japón. Al igual que los casos anteriores, el igualador no se encuentra especificado, siendo su esquema muy parecido al de IS-54 e IS-136.

[2] Del ingles “United States Digital Cellular”.

### 3.2.1 Métodos de igualación no lineal

En este apartado vamos a comentar los instrumentos utilizados en la actualidad en el diseño e implementación de métodos de igualación no lineal. Nuestro objetivo es el de detallar los métodos brevemente, y relacionarlos con referencias prácticas que enlacen con verdaderos desarrollos significativos que se estén estudiando e implementando en la actualidad.

#### 3.2.1.1 Redes neuronales

Una red neuronal es un procesador distribuido paralelo y masivo realizado con unidades simples de procesamiento, las cuales tienen especial predisposición natural en guardar conocimiento experimental, y ponerlo en disponibilidad para ser usado. Se comporta de manera semejante al cerebro en los siguientes aspectos [Haykin, 2005]:

- El conocimiento es obtenido por la red desde su entorno a través de un proceso de aprendizaje.
- Las conexiones interneuronales, conocidas como pesos sinápticos, son usadas para guardar el conocimiento adquirido.

Las redes neuronales tienen grandes beneficios, entendidos como capacidades y propiedades muy útiles, como por ejemplo:

- No linealidad: Una red neuronal puede ser lineal o no lineal, dependiendo de las neuronas de las que se encuentre formada. Es el camino, por el que vamos a intentar resolver nuestro problema de igualación de canal no lineal.
- Mapeo de entrada-salida: Está muy relacionado con el concepto de aprendizaje supervisado, a través del cual modificamos los pesos sinápticos de una red neuronal, realizando un aprendizaje a partir de unas muestras de entrenamiento. Cada muestra de entrenamiento enlaza una señal de entrada y la salida deseada. De esta manera, tenemos que nuestra red aprende a partir de los ejemplos, realizando un mapeo entre

las entradas y las salidas deseadas, con la única modificación de sus pesos sinápticos.

- Adaptativo: Derivado de lo anterior, podemos notar que la red neuronal es capaz de adaptar sus pesos sinápticos frente a cambios del entorno que le rodea.
- Respuesta evidente: Las redes neuronales nos dan la posibilidad de filtrar patrones ambiguos y centrarnos en lo particular del problema.
- Información contextual: La información contextual es tratada de forma natural por una red neuronal, ya que cada neurona de la red es afectada de manera potencial por el comportamiento del resto de neuronas.
- Tolerancia de fallos: Una red neuronal muestra una degradación suave en prestaciones, más que un fallo catastrófico, en el caso de errores o caída de conexiones entre neuronas.
- Implementabilidad VLSI[3]: Debido a la naturaleza principalmente paralela de la red neuronal, proporciona una gran rapidez en la computación de determinadas tareas, así como una facilidad para la implementación con tecnología VLSI.

Una vez comentadas las ventajas del procesado neuronal, podemos entrar en detalle en las neuronas. Entendemos una neurona como la unidad mínima de proceso de una red neuronal. El esquema básico de una neurona lo podemos ver en la Figura 3.1.

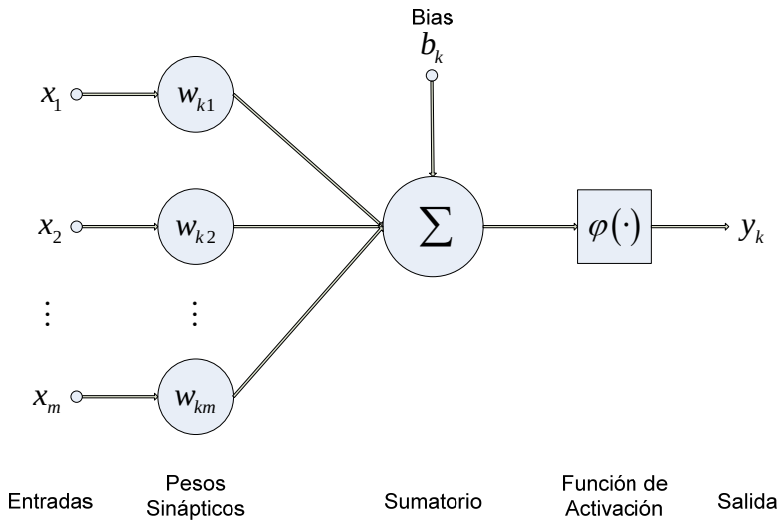



**Figura 3.1 – Esquema básico de una neurona.**

[3] Del inglés "Very Large Scale Integrated".

Normalmente el rango de salida de la neurona suele estar comprendido entre [0,1], o [-1,1]. La entrada "Bias" que tiene el diagrama, es opcional, y tiene el efecto de incrementar o disminuir la entrada de la función de activación. Escribiéndolo de manera matemática, tenemos que el comportamiento de una neurona viene gobernado por la expresión 3.1, donde $x_j$ corresponde con las entradas a la neurona, $w_{kj}$ con los pesos sinápticos, $u_k$ es la combinación lineal de salida, $b_k$ es la alimentación externa, y $\varphi(\bullet)$ es la función de activación.

$$\begin{aligned} u_k &= \sum_{j=1}^{m} w_{kj} x_j \\ y_k &= \varphi(u_k + b_k) \\ v_k &= u_k + b_k \end{aligned} \qquad \textbf{3.1}$$

Podemos utilizar diferentes tipos de funciones de activación, entre las que podemos encontrar por ejemplo:

- Función umbral: Se define en la expresión 3.2 y lo único que realiza es una decisión umbral fijo en el $v_k = 0$.

$$\varphi(v) = \begin{Bmatrix} 1 & ; & v \geq 0 \\ 0 & ; & v < 0 \end{Bmatrix} \qquad \textbf{3.2}$$

- Función lineal: Se define en la expresión 3.3 y utiliza una función lineal para obtener la relación entre $v_k$ y $y_k$.

$$\varphi(v) = \begin{Bmatrix} 1 & ; & v \geq \frac{1}{2} \\ v & ; & \frac{1}{2} > v > -\frac{1}{2} \\ 0 & ; & v \leq -\frac{1}{2} \end{Bmatrix} \qquad \textbf{3.3}$$

- Función sigmoide: Es una función más suave que las anteriores, y que recibe un parámetro 'a' que controla la pendiente. Su ecuación se muestra en la expresión 3.4.

$$\varphi(v) = \frac{1}{1+e^{(-av)}} \qquad \textbf{3.4}$$

Además de disponer de diferentes funciones de activación, las redes neuronales también se pueden clasificar en función de la estructura en la que se organizan las neuronas, estando muy relacionada la arquitectura de la red con el algoritmo de aprendizaje utilizado para entrenarla. Entre las posibles arquitecturas de una red neuronal, tenemos:

- Redes de una sola capa: Su esquema aparece en la Figura 3.2, y como se puede ver, las neuronas se organizan en capas. Tenemos una capa de nodos de entrada que alimentan nuestras neuronas, las cuales se encuentran en la capa de salida.

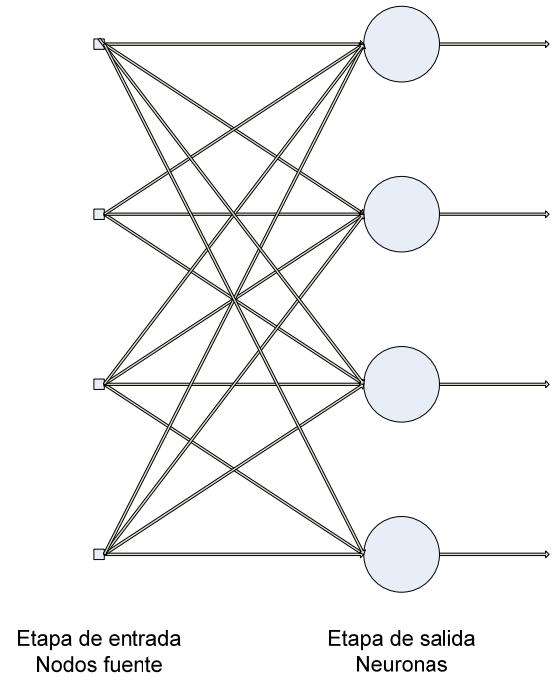



**Figura 3.2 – Red neuronal de una sola capa.**

- Redes multicapa: Podemos ver su arquitectura en la Figura 3.3 y se diferencia del caso anterior en que disponemos de mas de una capa de procesamiento, es decir, no solo tenemos la capa de neuronas a la salida. En este caso las capas de neuronas intermedias, se denominan capas ocultas, y su función es la de intervenir de alguna forma entre la entrada externa y la salida de la red. Al introducir nuevas neuronas y incrementar el número de enlaces sinápticos, la red puede trabajar con un significado mas

global y es capaz de llegar a mayores potencias, con lo que ganamos en precisión.

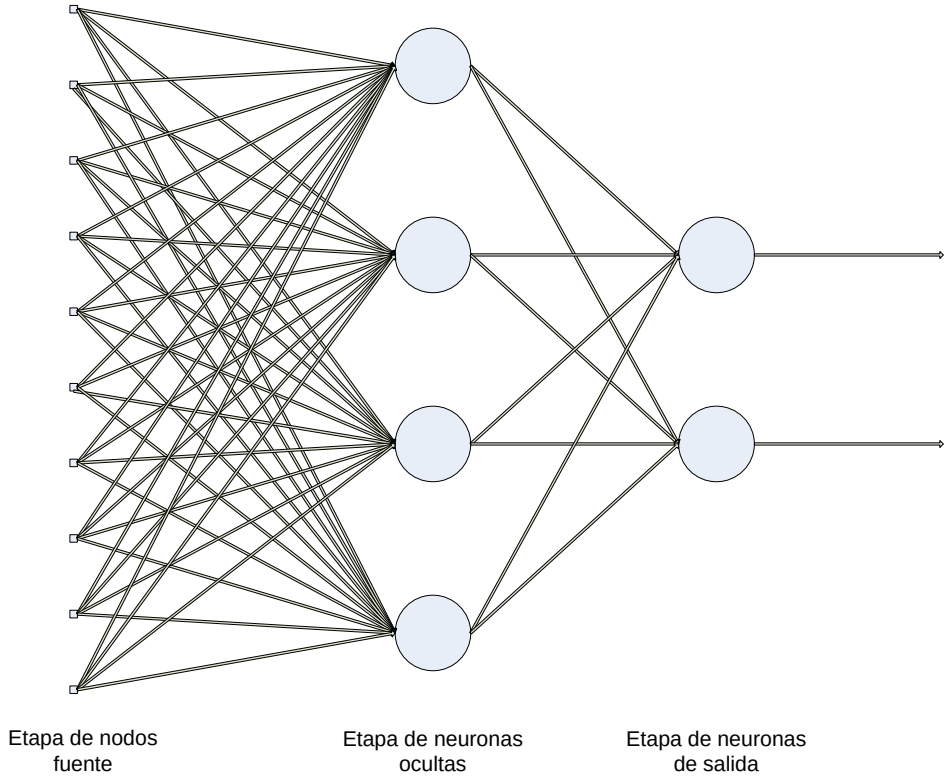



**Figura 3.3 – Red neuronal multicapa.**

- Redes recurrentes: Como vemos en la Figura 3.4, podemos notar que la diferencia principal de este tipo de redes es que poseen realimentación de su salida hacia su entrada. La realimentación produce un gran impacto en el entrenamiento de la red, y si introducimos retardos, la estamos haciendo no lineal.

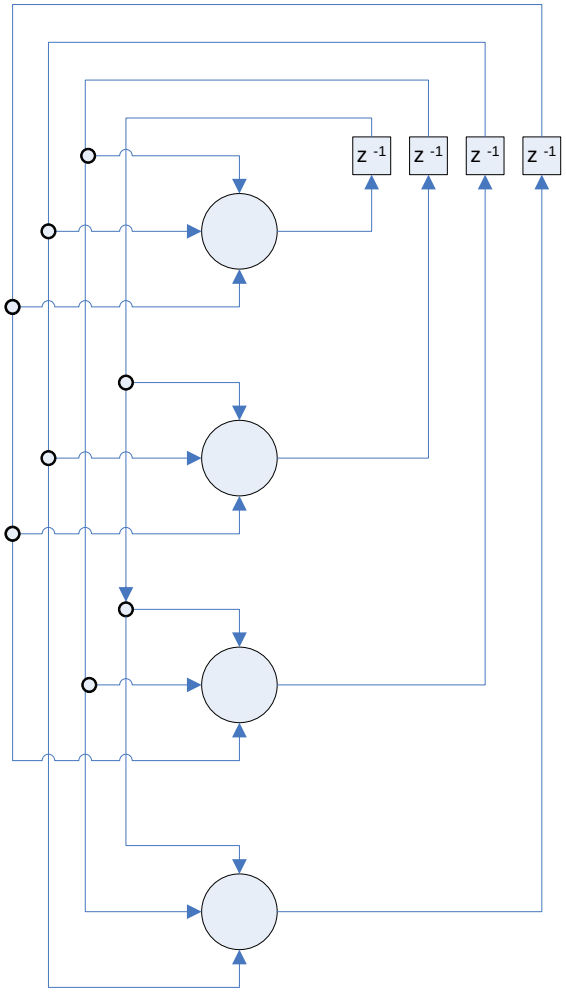



**Figura 3.4 – Red neuronal recurrente.**

En definitiva, tenemos que los tipos de arquitecturas comentadas son las formas básicas, y en cualquier caso, son perfectamente cohabitables, es decir, por ejemplo, se puede construir una red neuronal recurrente multicapa.

Las redes neuronales han sido propuestas para los problemas de igualación y detección en canales de comunicaciones, y en otras aplicaciones[4] no lineales de procesamiento de datos, entre los que podemos encontrar por ejemplo [G.J. Gibson, S. Siu, F.N. Cowan; 1989], [S.H. Bang, B.J. Sheu;1992], [S. Siu, G.J. Gibson, C. F. N. Cowan; 1990], [G. de Veciana, A. Zakhor; 1992], [S.K. Nair, J. Moon; 1994], [P. Lee;1996], [S. K. Nair, J. Moon; 1995].

Para entrar de manera más directa en algún ejemplo práctico, podemos mencionar el documento [S. K. Nair, J. Moon; 1997] en el que se aborda un

[4] Otro campo práctico muy amplio, aparte de la igualación de canal, en el que se emplean métodos no lineales, es el almacenamiento de datos sobre soportes magnéticos u ópticos.

estudio sobre la aplicación de técnicas de redes neuronales para su aplicación en problemas de igualación bajo canales de almacenamiento de datos magnéticos no lineales. El objetivo del documento es el de presentar métodos para obtener los pesos de las neuronas de las capas ocultas en un detector de señal basado en redes neuronales. El motivo de este estudio es debido a la falta de mecanismos que permiten diseñar las fronteras de decisión en este tipo de detectores a través del manejo de las capas ocultas y sus pesos. El documento comenta dos métodos, el primero de ellos maximizando la relación SNR, y el segundo de ellos minimizando la probabilidad de traspasar la frontera de decisión[5], asumiendo que el ruido de entrada se corresponde con una distribución Gaussiana.

Los resultados obtenidos en este ejemplo práctico comienzan con la afirmación de que los igualadores basados en redes neuronales siempre tienen una mayor relación señal a distorsión que los lineales. En el caso de perdida de información, las redes neuronales siguen funcionando mejor que su contrapunto lineal. Aparte del caso de pérdida de información, también es evidente el mejor funcionamiento del igualador basado en redes neuronales en situaciones con presencia de jitter[6] y variación de ruido, tanto en varianza como en distribución. Por último, y para comprobar el funcionamiento de los dos métodos comentados en el empleo de redes neuronales, los autores demuestran que las fronteras de decisión creadas con ambos métodos son muy cercanas a una red entrenada con el algoritmo de propagación hacia atrás[7] [D.D. Lincoll;1986], que es el procedimiento general de entrenamiento en el caso de redes neuronales con alimentación de datos hacia delante. En este tipo de redes se preprocesa la señal recibida antes de que se haya realizado ninguna decisión dura.

[5] Lo cual es equivalente a minimizar el criterio de probabilidad de error.
[6] Efecto que aparece al tener una dispersión en el retardo de las muestras de entrada.
[7] Algoritmo “back propagation”.

#### *3.2.1.1.1 Red de funciones de base radial*

Una red de funciones de base radial, del inglés Radial Basis Function (RBF), no es mas que una red neuronal supervisada, que utiliza la transformación del espacio de entrada (usualmente no separable) a un espacio de salida, que generalmente suele tener mas dimensiones que el de entrada, y en el que las clases de entrada son separables[8], ya que tenemos en este caso una superficie multidimensional que produce un mejor ajuste. Generalmente son usadas para problemas de interpolación y separación de patrones.

La construcción de una RBF tiene una arquitectura de una red neuronal de tres capas, cada una de las cuales con una utilidad completamente diferente. Las neuronas de la capa de entrada cumplen la función de nodos fuente, que conectan la red con su entorno. La segunda capa, que es la única capa oculta, tiene como función aplicar una función no lineal entre el espacio de entrada hacia el espacio oculto (realiza el "Kernel Trick"). Normalmente este espacio oculto suele ser de grandes dimensiones. La última capa es la capa de salida, y es lineal, proporcionando la respuesta de la red al patrón de activación que recibimos en la capa de entrada.

Bajo el contexto de redes neuronales, tenemos que las neuronas ocultas ofrecen un conjunto de bases para los patrones de entrada que tenemos en los nodos fuente. Estas bases son las que expanden las entradas en el espacio oculto, y reciben el nombre de "radial basis functions".

La explicación matemática del uso de una transformación no lineal y posteriormente una lineal, viene detallada en [Cover, 1965], y viene a decir que en un problema de clasificación de patrones, un espacio de grandes dimensiones en el cual se usen funciones no lineales, es más probable que tenga sus clases separables que otro espacio con bajas dimensiones. Además debemos notar que a mayor dimensión tengamos en la capa oculta, mayor precisión tendremos en el mapeo de entrada a salida de la red neuronal.

[8] Se conoce como "Kernel Trick", ya que aplicando una función a las vectores de un espacio de entrada, en el que sus patrones no son separables, nos movemos hacia un espacio de salida multidimensional en el que si son separables.

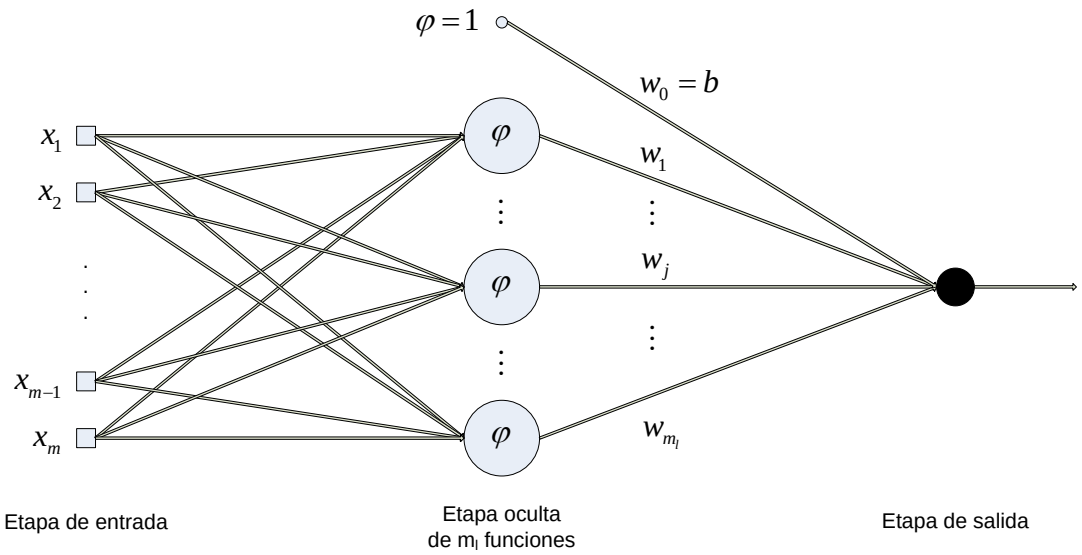



**Figura 3.5 – Red Radial Basis Function.**

Algunos ejemplos prácticos del uso de redes neuronales RBF en problemas de igualación de canal no lineal, son por ejemplo: [T.Junxia, DU. Liping, K. Jingming, W. Hua; 2004], así como sus referencias [I. Cha, S.A. Kassam; 1995], [J.Lee, C. Beach, N. Tepedelenlioglu;1999], [S. Chen, B. Mulgrew, P.M. Grant;1993].

En el documento principal [T.Junxia, DU. Liping, K. Jingming, W. Hua; 2004], tenemos que se propone un método de igualación no lineal adaptativo y que opera de forma online mediante el uso de redes neuronales RBF. Se justifica el uso de este tipo de redes neuronales porque tienen la habilidad de igualar canales no lineales. El método propuesto, en particular se basa en la aplicación de la técnica de gradiente estocástico, en la que se obtiene unas buenas prestaciones ajustando únicamente un coeficiente, y por lo tanto un único centro, el más cercano al vector de entrada.

El motivo de la elección de redes neuronales en la solución del documento que comentamos, es porque son capaces de aprender y autoorganizarse, y en particular, las RBFs, se emplean porque tienen una estructura similar a la Bayesiana óptima con decisión símbolo a símbolo, junto con que su diseño solo tiene dos capas, lo cual las hace ser más sencillas que su caso general y por lo tanto más fácilmente implementables. Finalmente la idea del algoritmo es el ajuste adaptativo y online del igualador con el objetivo de seguir variaciones lentas del

entorno. Los resultados que muestran los autores demuestran que el método empleado parece ser eficiente, aunque no lo hemos podido corroborar de forma práctica.

Existen una multitud de referencias de usos de redes neuronales RBF en el entorno no lineal. Entre ellas, métodos que unifican varias de las técnicas empleadas para hacer frente a la no linealidad, como por ejemplo [S. Chen, B. Mulgrew, P.M. Grant;1993] en la que se emplea técnicas de clasificación junto con redes RBF en el problema de igualación de canal. En detalle, se emplea un algoritmo de agrupamiento supervisado para entrenar la red RBF. Otro ejemplo de mezcla de técnicas puede ser [S. Bouchired, M. Ibnkahla, D. Roviras, F. Castaniè; 1998] en la que se emplean conjuntamente esquemas RBFs y SOMs[9].

Más ejemplos del uso de redes RBF en el ámbito que nos ocupa, los podemos obtener del uso de redes RBF mínimas (definidas en [L. Yingwei, N. Sundararajan, P. Saratchandran; 1996]) las cuales usan un esquema que les permite eliminar o añadir centros a la red, consiguiendo de esta forma llegar a una red de tamaño mínimo. Tenemos los casos de [P.C. Kumar, P. Saratchandran, N. Sundararajan;1998] y [P.C. Kumar, P. Saratchandran, N. Sundararajan; 2000].

También podemos encontrar ejemplos de aplicación de las redes RBF en el problema de igualación no lineal, junto con canales variantes, que es la situación en la que realizaremos nuestro estudio final en el Capítulo 5, como por ejemplo [R. Assaf, S. El Assad, Y. Harkouss] ó [M. Mimura, N. Hamada, T. Furukawa].

[9] Mapas Auto Organizados.

### 3.2.1.2 Maquinas de vectores soporte

Las Máquinas de Vectores Soporte (SVM) se catalogan como métodos de aprendizaje máquina supervisado, y se utilizan en clasificación y regresión principalmente. Tienen como comportamiento característico maximizar la distancia entre la frontera de decisión (hiperplano de separación) y las muestras más cercanas.

Para poder construir el hiperplano de separación, las SVMs pueden transformar el espacio de entrada en uno de mayor dimensión, donde se espera que los datos de entrada sean linealmente separables, o en su defecto que se aumente la separación entre las clases existentes.

Podemos distinguir dos tipos de SVMs en función de la estructura interna de la máquina. De esta manera las clasificamos en SVMs lineales y su variante no lineal, siendo las primeras más sencillas.

#### 3.2.1.2.1 SVM lineal

Partimos del supuesto de que tenemos unos datos de entrada, destinados para el entrenamiento de la máquina, y etiquetados en pares de la forma $\{x_i, y_i\}$. Suponiendo un caso binario en el que $y_i$ solo pueda contener los valores [-1, 1], y que las dos clases son linealmente separables, el hiperplano que las separaría, cumplirá: $w^T x + b = 0$, donde $w$ es el vector normal al hiperplano y $|b|/\|w\|$ es la distancia del hiperplano al origen de coordenadas. Suponiendo también que denotamos la distancia del hiperplano a las muestras de cada clase mas cercanas como $d^+$ y $d^-$, tenemos que el margen que tiene la SVM será $d^+ + d^-$.

Para encontrar el mejor hiperplano con mejor margen, tenemos que colocar nuestro hiperplano en la mitad de la distancia existente entre los patrones de entrenamiento de clases distintas más cercanas entre sí. A estas muestras que nos sirven para determinar en que posición colocamos el hiperplano las denominamos vectores soporte.

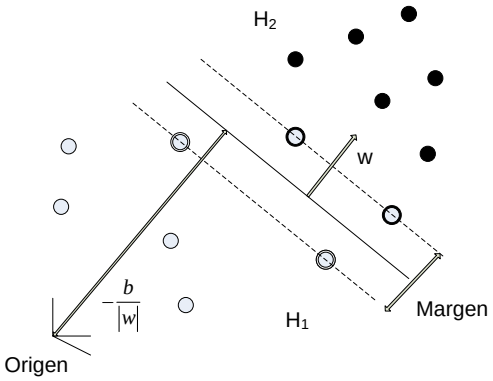



**Figura 3.6 – Hiperplano separador de las SVMs.**

### 3.2.1.2.2 SVM no lineal

Cuando las clases no son linealmente separables, el uso de la SVM lineal no nos lleva a los mejores resultados. La mejora que introducimos con respecto al caso anterior es el uso de funciones Kernel que suponen de forma implícita una transformación del espacio de entrada a otro de dimensión mayor[10], donde se espera que las clases si sean linealmente separables, o que la distancia entre ellas aumente.

En la definición del hiperplano en SVMs, los datos de entrenamiento aparecen en forma de producto escalar, $x_i^T x_j$, así que si transformamos los datos de entrada a un espacio Euclideo $\mathrm{H}$ de mayor dimensión usando la función $\varphi(\cdot)$, tenemos que $\varphi: R^d \to \mathrm{H}$.

En consecuencia a lo dicho anteriormente, tendremos que el problema del entrenamiento dependerá de $\varphi^T(x_i)\cdot\varphi(x_j)$ en el espacio $\mathrm{H}$. Si encontramos una función Kernel que realice la operación $K(x_i, x_j) = \varphi^T(x_i)\cdot\varphi(x_j)$, podríamos usarla directamente sin necesidad de conocer $\varphi$ a priori.

Una vez elegido el Kernel a usar, solo hay que cambiar la expresión de entrenamiento de la SVM, que nos devuelve la ecuación del hiperplano y que

[10] Empleamos el "Kernel Trick" de nuevo.

depende de $x_i^T x_j$, por $K\left(x_i, x_j\right)$, y continuar exactamente igual con la derivación del hiperplano.

| Kernel | Función |
|---|---|
| Lineal | $K\left(x_i, x_j\right) = \left(x_i^T \cdot x_j\right)$ |
| Polinómico | $K\left(x_i, x_j\right) = \left(x_i^T \cdot x_j + c\right)^p$ |
| RBF[11] | $K\left(x_i, x_j\right) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}\right)$ |
| Sigmoidal | $K\left(x_i, x_j\right) = \tanh\left(kx_i^T \cdot x_j - \delta\right)$ |

**Tabla 3.1 – Funciones Kernel mas comunes.**

Podemos encontrar ejemplos prácticos del empleo de las SVMs en el campo de la igualación de canal, entre los que se encuentran el estudio que hace de ellas en [D.J. Sebald, J.A. Bucklew; 2000] sobre igualación no lineal, en la que justifica su uso debido a la similitud de prestaciones que existe entre las SVMs y las redes neuronales en este campo de estudio.

También podemos encontrar ejemplos del uso de las SVMs en otros campos de trabajo, que aunque no son el campo principal en el que enmarcamos nuestro trabajo, si son aplicaciones sobre un entorno no lineal, con lo que en consecuencia, serían fácilmente extrapolables al campo de la igualación de canal no lineal. En este caso nos encontramos frente al documento [L. Ramamoorthy, M. Keshinoz, B.V.K.V. Kumar; 2005] que aplica las SVMs en la aplicación de almacenaje en sistemas de datos holográficos. Otro caso [N. Chithra Raj, P.V. Aswathy, K.V. Sagar; 2007] en el que el campo de aplicación de las SVM no lineales corresponde con el estimador del ángulo de llegada en sistemas de comunicaciones CDMA.

[11] En este caso, el espacio $H$ asociado tiene dimensión infinita, por lo que trabajar directamente con $\varphi$ no es posible a efectos prácticos.

### 3.2.1.3 Mapas Auto Organizados

Los Mapas Auto Organizados (SOM) diseñados por Kohonen [Kohonen; 1982] son un tipo mas de redes neuronales artificiales, basadas en aprendizaje competitivo. Este tipo de aprendizaje se caracteriza porque las neuronas de salida de la red compiten entre ellas para estar activadas o desactivadas, ya que en cualquier caso, solo una única neurona de la etapa de salida o de un grupo puede estar encendida en un instante de tiempo. A la neurona que vence a las demás se le denomina neurona ganadora.

En un SOM las neuronas suelen colocarse en los nodos de una malla que normalmente suele tener una o dos dimensiones.

El objetivo principal de los SOMs es transformar un patrón de entrada de cualquier dimensión en un mapa de una o dos dimensiones discreto, de forma adaptativa. De esta forma, tenemos que un SOM está caracterizado por la formación de un mapa topográfico de los patrones de entrada en el que las coordenadas espaciales de las neuronas en la red son indicativas de las características intrínsecas contenidas en los patrones de entrada.

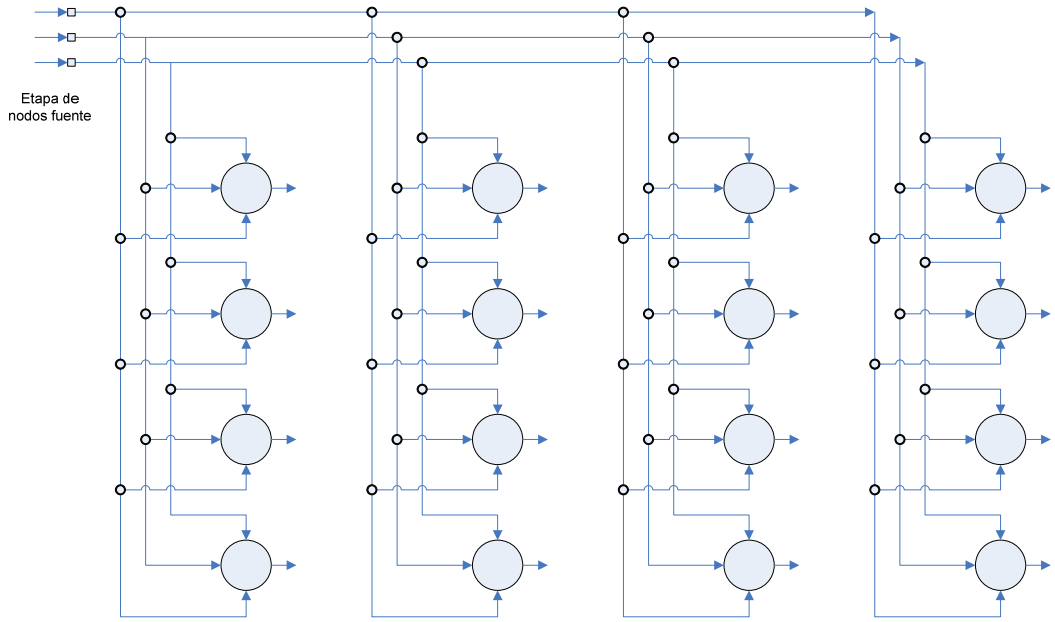



**Figura 3.7 – Malla de neuronas bidimensional.**

Tal y como se puede notar en la Figura 3.7, cada neurona de la malla está completamente conectada a todos los nodos fuente de la etapa de entrada. Al introducir un patrón de entrada, encontramos que aparece una zona localizada de actividad. La naturaleza y localización de esta zona varía de una realización de entrada a otra, con lo que debemos asegurarnos que todas las neuronas de la red hayan sido expuestas a los suficientes patrones de entrada, para garantizar que el proceso de auto organización se realiza correctamente.

El algoritmo encargado de la formación de los mapas auto organizados inicializa primero los pesos sinápticos de la red. Una vez hecho esto, se utilizan 3 procesos fundamentales:

- Competición: Para cada patrón de entrada, las neuronas de la red calculan sus respectivos valores usando una función discriminante. Esta función la que crea la base sobre la que las neuronas compiten entre sí. La neurona que obtenga el valor mayor en la función discriminante será la denominada neurona ganadora.

- Cooperación: La neurona ganadora determina la localización espacial de las neuronas vecinas topologicamente, proporcionando de esta manera una base de cooperación entre dichas neuronas vecinas.

- Adaptación sináptica: Este mecanismo habilita a las neuronas excitadas la capacidad de incrementar sus valores individuales de la función discriminante en relación con el patrón de entrada, a través de modificaciones en sus pesos sinápticos. Este ajuste es la respuesta de la neurona ganadora ante patrones posteriores similares.

Podemos encontrar ejemplos de uso de las SOM sobre nuestro problema de igualación de canal, como por ejemplo [W. Paquier, M. Ibnkahla; 1998] en el cual se usan las SOM en un esquema de igualación no lineal con rápido desvanecimiento.

También son muy empleadas conjuntamente con las redes neuronales, como por ejemplo [X. Wang, J. Hua Lin, N. Zhang, H. Sekiya, T. Yahagi; 2002] en la que se utilizan con redes neuronales recurrentes, o [S. Bouchired, M. Ibnkahla, D. Roviras, F. Castanié; 1998] en la que se aplican con redes RBF, o [M. Ibnkahla,

F. Castanie; 1995] que emplean las SOM en un esquema conjunto con redes neuronales vectoriales.

### 3.2.1.4 Series de Volterra

Otro mecanismo empleado en la igualación no lineal, es el uso de las series de Volterra, ya que son consideradas un modelo de comportamiento no lineal. Son similares a las Series de Taylor, con la salvedad de que estas tienen efectos de memoria. La teoría de Volterra es una generalización de la convolución lineal aplicada en sistemas LIT[12], y viene a decirnos que cualquier sistema no lineal invariante en el tiempo puede ser modelado como una suma infinita multidimensional de integrales de convolución de orden ascendente (expresión 3.5 en continuo y 3.6 en discreto).

$$y(t) = h_0 + \int_0^\infty h_1(\tau_1)u(t-\tau_1)d\tau_1 + \int_0^\infty \int_0^\infty h_2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2)d\tau_1 d\tau_2 + \ldots \quad \textbf{3.5}$$

$$y[n] = h_0 + \sum_{k=0}^{N} h_1[n-k]u[k] + \sum_{k_1=0}^{N}\sum_{k_2=0}^{N} h_2[n-k_1, n-k_2]u[k_1]u[k_2] + \ldots \quad \textbf{3.6}$$

En la expresión 3.5/3.6, tenemos que $u(t)/u[n]$ representa la entrada del sistema dinámico, mientras que $y(t)/y[n]$ representa la respuesta del sistema. Cada suma de convolución contiene un kernel, $h_n$ que es utilizado para caracterizar el comportamiento del sistema. El conocimiento de estos coeficientes permite la predicción de la respuesta del sistema ante cualquier entrada arbitraria. Podemos notar que el primer elemento que aparece en la serie de Volterra corresponde con la convolución lineal, mientras que la no linealidad la introducimos en los coeficientes de kernel de orden superior. El sentido del uso de los kernels de Volterra radica en el hecho de que cada uno de los coeficientes está relacionado con distintos instantes de tiempo de la entrada, con lo que tenemos representadas las respuestas previas llevadas a través de nuestro sistema.

En particular para el uso de las series de Volterra en el problema de igualación no lineal, disponemos de varios mecanismos matemáticos, como por ejemplo el teorema del punto fijo (no lo detallamos, porque lo haremos más adelante en el siguiente capítulo). Con este tipo de mecanismos, junto con el apoyo del modelo no lineal de la serie, podemos llegar a ejemplos tales como, [R.D. Nowak, B.D. Van Veen; 1997] en el que emplea un mecanismo de punto fijo

[12] Siglas de "Sistema Lineal Invariante".

para resolver el problema de igualación no lineal[13]. Otra solución del algoritmo de Volterra empleada en nuestro problema puede verse en [K. Meghriche, S. Femmam, B. Derras, M. Haddadi; 2001] en el que se presenta un algoritmo recursivo que sigue las variaciones en el tiempo, actualizando los coeficientes lineal y cuadrático de un filtro de Volterra. Otra alternativa es la que se presenta en [A.J. Redfern, G.T. Zhou; 1998] en la que se emplea un nuevo método de igualación basado en Volterra cuyo objetivo es el de encontrar las raíces del polinomio de la entrada[14]. También podemos destacar otro método de igualación empleado [M.T. Özden, A.H. Kayran, E. Panayirci; 1998] en el que se construye un igualador basado en expansión en serie de Volterra, trasformando el problema de igualación no lineal en otro equivalente pero multicanal y lineal. De esta manera tenemos la entrada no lineal completamente ortogonalizada utilizando etapas de procesamiento multicanal en celosía.

Además del campo de aplicación de igualación de canal, existen otros muchos en los que se emplea la serie de Volterra, como por ejemplo [J.N. Lin, R. Unbehauen; 1992] en el que nos encontramos con la aplicación de un filtro 2D de Volterra implementado con un algoritmo LMS para aplicación en igualación de canal no lineal, y restauración de imágenes. Otro ejemplo de este tipo puede verse en [L.Agarossi, S. Bellini, F. Bregoli, P, Migliorati; 1998] en el que se realiza una comparación de algoritmos empleados en la igualación sobre canales ópticos no lineales. En esta comparación aparece en escena el algoritmo NAVE[15], el cual no es más que un igualador adaptativo no lineal basado en series de Volterra.

Para finalizar el estudio de los documentos relacionados con las series de Volterra y empleados en aplicaciones no lineales, vamos a comentar referencias en las cuales obtenemos un estudio base de la convergencia de este tipo de métodos. Consideramos que esto es importante, ya que la serie de Volterra es infinita y con coeficientes de alto grado, lo cual puede hacer tender la serie a la inestabilidad muy fácilmente. Podemos citar [Y.-W. Fang, L.-Ch. Jiao, X.-D. Zhang, J. Pan; 2001] en el que se examina la estabilidad local de los igualadores de Volterra basados en la inversa de orden-p del sistema bajo estudio.

[13] Este mecanismo utilizado en este documento, es semejante al ejemplo que comentaremos en el epígrafe 4.3 y que construiremos en el Capítulo 5. No obstante la implementación que realizaremos no corresponde al artículo comentado en este punto.
[14] Se presupone que la entrada posee un alfabeto finito de símbolos.
[15] Del ingles “Nonlinear Adaptive Volterra Equalizer”.

# Capítulo 4 Algoritmos implementados

## *4.1 Introducción*

En este cuarto capítulo vamos a hacer un recorrido por los algoritmos que hemos implementado de forma práctica en el desarrollo de nuestro Proyecto Fin de Carrera. No debemos olvidar la verdadera finalidad que hemos tenido desde el principio del presente documento, así como el objetivo de éste, que no es mas que obtener una comparación lo más general posible en el marco de igualación de canal no lineal y variante en el tiempo, entre el algoritmo KRLS[1] y otros algoritmos de referencia en el campo de la igualación de canal no lineal adaptativa. Por ese motivo hemos comentado en el capítulo anterior las técnicas mas empleadas en este tipo de problemas.

Según lo dicho, organizamos este capítulo de la forma más coherente posible, es decir, comenzaremos por nuestro algoritmo a comparar, KRLS, y continuaremos detallando cada uno de los algoritmos que hemos utilizado para comparar las prestaciones del primero, dentro del amplio abanico comentado en el capítulo anterior.

No introduciremos entre los algoritmos comentados ninguno relacionado con el filtrado de Kalman (tal y como contamos en el epígrafe 2.4 es la solución óptima en un entorno de igualación con canal variante), ya que está centrado en canales variantes lineales, y posee parámetros difíciles de estimar en su modelo, como las matrices de covarianza de ruido $\underline{\underline{R}}$ y $\underline{\underline{Q}}$. Estos aspectos no cumplen

[1] Del inglés “Kernel Recursive Least Squares”.

correctamente con las especificaciones de nuestro problema de igualación no lineal variante y online (al tener parámetros difíciles de estimar, incurrimos en retrasos en el procesamiento). Por esta razón se eligió como alternativa lineal en la comparativa el algoritmo LMS, en lugar del filtro de Kalman.

Hemos de mencionar que en el presente capítulo, no tenemos como objetivo la comprobación de los algoritmos que mostraremos de cada uno de los métodos, ni del KRLS en particular, ya que estos resultados se mostrarán y se analizarán con detalle en el capítulo siguiente. Nos limitaremos a detallar cada algoritmo.

## *4.2 Kernel Recursive Least Squares*

Comenzamos nuestro recorrido por la implementación del algoritmo en el cual basamos nuestro Proyecto Fin de Carrera fundamentado en el documento de referencia [Y. Engel, S. Mannor, R. Meir;2004] con título “The Kernel Recursive Least-Squares Algorithm” (KRLS). Comentaremos en detalle este algoritmo, ya que el objetivo es medir sus prestaciones comparándolas frente a otros que empleen métodos de igualación no lineal de canal alternativos.

El algoritmo KRLS presenta una versión no lineal del algoritmo RLS comentado en el apartado 2.3.1.1, el cual, como sabemos, busca la minimización de los errores cuadráticos (MSE).

Para conseguir la no linealidad del algoritmo lineal, se utilizan las funciones Kernel Mercer, los cuales se aplican sobre los vectores de entradas, obteniendo un resultado que se interpreta como un producto en un espacio de Hilbert de elevadas dimensiones[2]. De esta manera podemos hacer productos internos sobre el espacio característico de Hilbert en grandes dimensiones sin hacer referencia a los vectores de entrada del espacio origen.

Esta herramienta puede ser usada sobre cualquier algoritmo lineal, aunque nos centraremos en el uso sobre el algoritmo RLS, ya que es un algoritmo muy conocido y estudiado, además de eficiente en la predicción lineal de mínimos cuadrados.

Otro aspecto que determina el estudio del algoritmo es la funcionalidad online. Lo que se persigue es que el estimador sea modificado según van llegando muestras. Si además queremos que el algoritmo opere en tiempo real, debemos añadir el requisito de que el coste computacional por muestra no se incremente en función del tamaño del conjunto de datos.

[2] Al uso de las funciones Kernel para obtener productos en espacios de elevadas dimensiones, resolviendo de esta manera tener que realizar productos de las transformaciones de cada vector de entrada, que por otro lado puede no ser realizable, se le conoce como “Kernel Trick”, o truco de Kernel. Ya lo comentamos en el capítulo anterior.

En general la solución obtenida mediante el uso de kernels es no paramétrica y definida con la forma $\hat{f}(x) = \sum_{i=1}^{l} \alpha_i k(\underline{x}_i, \underline{x})$, donde $\alpha$ son los pesos que ponderan la función del kernel que se aplica sobre las muestras de entrenamiento $\{\underline{x}_i\}_{i=1}^{l}$.

Nuestro objetivo es conseguir una solución reducida, con la cual mediante el uso de una mínima parte de los vectores de entrada, seamos capaces de reconocer cualquier vector posible de entrada. Para conseguir este objetivo disponemos de dos enfoques: el primero de ellos se basa en la *eliminación*, en la que a partir de todas las muestras de entrenamiento, vamos reduciéndolas hasta conseguir los miembros de la expansión que producen la solución reducida. La segunda opción, que es la que persigue el algoritmo KRLS, es la *construcción* de la expansión según llegan las muestras de entrenamiento.

En principio ninguno de estos dos enfoques nos sirve puramente para nuestro algoritmo, ya que tenemos como objetivo que sea *online*. De esta manera en cada instante de tiempo, tendremos una única muestra de entrenamiento y una única decisión que realizar.

La solución que propone el algoritmo KRLS es un enfoque online constructivo de reducción. Lo que hacemos es admitir en nuestro kernel solo aquellas muestras que no pueden ser halladas como combinación lineal de las muestras que ya se encuentran dentro del kernel. Para discriminar esta decisión, se dispone de un margen valido sobre el cual realizaremos la elección.

### 4.2.1 Pasos previos del algoritmo

Para comenzar con el algoritmo, debemos saber antes unas cuantas cosas sobre el enfoque que hemos tomado para kernelizar nuestro algoritmo RLS.

Un kernel es una función de k: $X \times X \to \Re$ que se asume continua. En particular un kernel Mercer es definido positivo, lo que significa que ante un número finito de muestras $\{\underline{x}_1, ..., \underline{x}_n\}$ tendremos una matriz de aplicación $\left[\underline{\underline{K}}\right]_{ij} = k\left(\underline{x}_i, \underline{x}_j\right)$ definida positiva. Utilizando el teorema de Mercer, podemos decir que existe un espacio de Hilbert $\mathcal{H}$ y una aplicación $\phi$ tal que: $\mathcal{X} \to \mathcal{H}$ con $k(x, x') = \langle \phi(x), \phi(x') \rangle$ , en el que $\langle \cdot, \cdot \rangle$ se refiere al producto escalar en $\mathcal{H}$. Esto es lo que usamos para convertir nuestro algoritmo adaptativo RLS lineal a un RLS no lineal, a través del uso del kernel.

Con respecto al método de reducción online, debemos decir, que perseguimos la expresión 4.1. Esto sigue siendo un estimador lineal en $\mathcal{H}$. Pensando por un momento que podamos escribir la expresión 4.2, podríamos poner cualquier vector del espacio característico $\mathcal{H}$, como combinación lineal de los anteriores vectores del propio espacio característico.

Esto último es lo que se realiza en el algoritmo KRLS que vamos a estudiar, con la única diferencia que permitimos que la aproximación con la combinación lineal no sea totalmente exacta, sino que dependa de un parámetro $\nu$ que determina la precisión de la reducción.

$$\hat{f}(x) = \sum_{i=1}^{l} \alpha_i k\left(\underline{x}_i, \underline{x}\right) = \sum_{i=1}^{t} \alpha_i \left\langle \phi\left(\underline{x}_i\right), \phi\left(\underline{x}\right) \right\rangle \qquad \textbf{4.1}$$

$$\phi\left(\underline{x}_t\right) = \sum_{i=1}^{t-1} a_i \phi\left(\underline{x}_i\right) \qquad \textbf{4.2}$$

Primero debemos definir nuestro diccionario como podemos ver en la expresión 4.3, siendo $\tilde{x}_j$ entradas tomadas de las muestras de entrenamiento $\{(\underline{x}_1, y_1), (\underline{x}_2, y_2), ...\}$, linealmente independientes en $\mathcal{H}$.

$$\underline{\underline{D}}_{t-1} = \left\{ \underline{\tilde{x}}_j \right\}_{j=1}^{m_{t-1}} \quad \textbf{4.3}$$

El funcionamiento del parámetro $\nu$ es el siguiente: llega una nueva dupla de entrenamiento $(\underline{x}_t, y_t)$, tomamos la entrada y la introducimos en nuestra aplicación del kernel para llevarla a $\mathcal{H}$. Una vez hecho esto, tenemos $\phi(\underline{x}_t)$ y testeamos si es o no aproximadamente linealmente dependiente de los vectores de entrada que tenemos en el diccionario, mediante la expresión 4.4, que nos lleva a definir el parámetro $\delta$ en la expresión 4.5.

$$\delta_t \overset{def}{=} \min_{\underline{a}} \left\| \sum_{j=1}^{m_t - 1} a_j \phi\left(\underline{\tilde{x}}_j\right) - \phi\left(\underline{x}_t\right) \right\|^2 \leq \upsilon \quad \textbf{4.4}$$

$$\delta_t = \min_{\underline{a}} \left\{ \underline{a}^T \underline{\underline{\tilde{K}}}_{t-1} \underline{a} - 2\underline{a}^T \underline{\tilde{k}}_{t-1}(x_t) + k_{tt} \right\} \quad \textbf{4.5}$$

De las expresiones 4.4 y 4.5 podemos definir las operaciones asociadas al kernel y mostradas en la expresión 4.6 , donde $k(\cdot,\cdot)$ denota la operación del kernel.

$$\begin{gathered} \left[ \underline{\underline{\tilde{K}}}_{t-1} \right]_{i,j} = k\left(\tilde{x}_i, \tilde{x}_j\right) \\ \left( \underline{\tilde{k}}_{t-1}(x_t) \right)_i = k\left(\tilde{x}_i, x_t\right) \\ k_{tt} = k\left(x_t, x_t\right) \end{gathered} \quad \textbf{4.6}$$

Si obtenemos $\delta_t \leq \upsilon$ consideramos la nueva entrada $\underline{x}_t$ linealmente dependiente de los vectores del diccionario en el espacio $\mathcal{H}$, con lo que buscamos los coeficientes $\underline{a} = \left(a_1, ...., a_{m_{t-1}}\right)^T$ de la combinación con la expresión 4.23 obtenida de la minimización de la expresión 4.7. Además el diccionario no se modificará.

$$\underline{a}_t = \tilde{\underline{\underline{K}}}_{t-1}^{-1} \tilde{\underline{k}}_{t-1}(x_t) \qquad \textbf{4.7}$$

Por el contrario si $\delta_t > \upsilon$ tendremos que ampliar el diccionario con la consiguiente modificación que puede verse en la ecuación 4.8.

$$\begin{aligned} D_t &= D_{t-1} \cup \{\underline{x}_t\} \\ m_t &= m_{t-1} + 1 \end{aligned} \qquad \textbf{4.8}$$

Con este mecanismo podemos asegurar que podemos poner todos los elementos de entrada a $\mathcal{H}$ como combinación lineal del diccionario con un cierto error, mostrado en 4.9, siendo en cualquier caso $\left\|\phi_i^{res}\right\|^2 \leq \nu$. O lo que es lo mismo en términos de matrices del kernel, podemos poner la matriz completa del kernel $\underline{\underline{K}}_t$ en función de la matriz reducida del kernel basada en el diccionario y que denotaremos por $\tilde{\underline{\underline{K}}}_t$, y cuya definición aparece en la expresión 4.10, con $\left[\underline{\underline{A}}_t^T\right]_{i,j} = a_{i,j}$.

$$\phi(\underline{x}_i) = \sum_{j=1}^{m_i} a_{i,j}\phi(\tilde{\underline{x}}_j) + \phi_i^{res} \qquad \textbf{4.9}$$

$$\underline{\underline{K}}_t \approx \underline{\underline{A}}_t \tilde{\underline{\underline{K}}}_t \underline{\underline{A}}_t^T \qquad \textbf{4.10}$$

### 4.2.2 Algoritmo KRLS

Una vez comentados los pasos previos estamos en disposición de formular las expresiones que definen nuestro algoritmo KRLS.

Nuestro objetivo es minimizar la expresión de coste mostrada en 4.11. Despejando nuestros pesos $\underline{w}$, llegamos a 4.12, y reescribiendo la función de coste en términos de nuestra matriz de kernel, $\underline{\underline{K}}$, obtenemos finalmente 4.13.

$$L(\underline{w}) = \sum_{i=1}^{t} \left( f(\underline{x}_i) - y_i \right)^2 = \left\| \underline{\underline{\Phi}}_t^T \underline{w} - \underline{y}_t \right\|^2 \quad \textbf{4.11}$$

$$\underline{w}_t = \sum_{i=1}^{t} \alpha_i \underline{\phi}(\underline{x}_i) = \underline{\underline{\Phi}}\,\underline{\alpha} \quad \textbf{4.12}$$

$$L(\underline{\alpha}) = \left\| \underline{\underline{K}}_t \underline{\alpha} - \underline{y}_t \right\|^2 \quad \textbf{4.13}$$

Ahora si en vez de usar el $\underline{\alpha}$ de 't' coeficientes derivados del uso de la matriz de kernel completa $\underline{\underline{K}}_t$, usamos un $\underline{\tilde{\alpha}}$ reducido de $m_t \times m_t$ coeficientes procedente del uso de la matriz de kernel reducida al diccionario, $\underline{\underline{\tilde{K}}}_t$, podemos llegar a la expresión 4.14 y reescribir nuestra función de coste en estos términos como mostramos en 4.15. Por último, minimizando la función de coste 4.15, podemos obtener una ecuación (expresión 4.16) que nos permite calcular $\underline{\tilde{\alpha}}_t$.

$$\underline{\tilde{\alpha}}_t = \underline{\underline{A}}_t^T \underline{\alpha}_t \quad \textbf{4.14}$$

$$L(\underline{\tilde{\alpha}}) = \left\| \underline{\underline{\Phi}}_t^T \underline{\underline{\tilde{\Phi}}}_t \underline{\tilde{\alpha}} - \underline{y}_t \right\|^2 = \left\| \underline{\underline{A}}_t \underline{\underline{\tilde{K}}}_t \underline{\tilde{\alpha}} - \underline{y}_t \right\|^2 \quad \textbf{4.15}$$

$$\underline{\tilde{\alpha}}_t = \underline{\underline{\tilde{K}}}_t^{-1} \left( \underline{\underline{A}}_t^T \underline{\underline{A}}_t \right)^{-1} \underline{\underline{A}}_t \underline{y}_t \quad \textbf{4.16}$$

Una vez obtenida la función de los pesos de nuestro KRLS, solo nos queda definir la matriz $\underline{\underline{P}}$ como se muestra en la expresión 4.17 y proceder a comentar los pasos necesarios para la implementación.

$$\underline{\underline{P}} = \left( \underline{\underline{A}}_t^T \underline{\underline{A}}_t \right)^{-1} \quad \textbf{4.17}$$

#### 4.2.2.1 Inicialización

Tomamos la primera muestra de la colección de entrenamiento $\{\underline{x}_1, y_1\}$ y calculamos los siguientes valores:

- Obtenemos el kernel del vector de entrada que tenemos:

$$k_{11} = k(\underline{x}_1, \underline{x}_1)$$

- Ya que solo tenemos este único vector, lo introducimos en nuestro diccionario sin comprobar el test de dependencia lineal. Aumentamos la longitud del diccionario, 'm', a 1 y introducimos el índice de entrada en el vector de índices, 'ii':

$$\underline{\underline{D}}_1 = [\underline{x}_1] \quad \Rightarrow \quad m_1 = 1 \quad \Rightarrow \quad ii = 1$$

- Calculamos tanto la matriz del kernel reducida sobre nuestro diccionario, como su inversa:

$$\underline{\underline{\tilde{K}}}_1 = [k_{11}] \quad \Rightarrow \quad \underline{\underline{\tilde{K}}}_1^{-1} = \left[\frac{1}{k_{11}}\right]$$

- Inicializamos los pesos de dependencia lineal, $\underline{a}$, así como la matriz de pesos de dependencia lineal del kernel reducido $\underline{\underline{A}}$:

$$\underline{a}_1 = [1] \quad \Rightarrow \quad \underline{\underline{A}}_1 = [\underline{a}]$$

- Inicializamos la matriz auxiliar $\underline{\underline{P}}$:

$$\underline{\underline{P}}_1 = [1]$$

#### 4.2.2.2 Paso 2: Bucle

Este paso lo repetimos mientras nos resten duplas de entrenamiento $\{\underline{x}_t, y_t\}$, comenzando en $t = 2$ hasta el fin de las muestras de entrenamiento en $t = M$.

i. Tomar nueva muestra: Tomamos nuestra nueva muestra de entrenamiento $\{\underline{x}_t, y_t\}$ y le aplicamos la función del kernel.

- $k_{tt} = k(\underline{x}_t, \underline{x}_t)$

ii. Calcular $\underline{\tilde{k}}_{t-1}(x_t)$: Una vez obtenido el resultado de aplicar el kernel sobre el nuevo vector de entrada, nos hemos desplazado del espacio de partida al espacio característico $\mathcal{H}$. En este espacio, podemos comparar nuestro resultado con la matriz del kernel reducida que también pertenece a este subespacio. Por esta razón calculamos $\underline{\tilde{k}}_{t-1}(x_t)$ aplicando la función del kernel sobre nuestra nueva entrada en 't' y sobre todos los elementos del diccionario que dispongamos hasta el instante anterior, 't-1'.

- $\underline{\tilde{k}}_{t-1}(\underline{x}_t) = k(\underline{\tilde{x}}_i, \underline{x}_t) \quad \forall i \in \{1,...,m_{t-1}\}$ [3]

iii. Test de dependencia lineal: Una vez obtenido el parecido de nuestro vector de entrada $\underline{x}_t$ con todos los vectores del diccionario hasta ese instante en el subespacio característico $\mathcal{H}$. estamos en disposición de plantear el test que determina si la nueva entrada es linealmente dependiente del diccionario con un cierto error $\nu$.

- $\underline{a}_t = \underline{\underline{\tilde{K}}}_{t-1}^{-1} \underline{\tilde{k}}_{t-1}(\underline{x}_t)$
- $\delta_t = k_{tt} - \underline{\tilde{k}}_{t-1}(\underline{x}_t)^T \underline{a}_t$
- Si $\delta_t > \nu$ se añade $\underline{x}_t$ al diccionario:

a. Actualizamos el diccionario:

$$\underline{D}_t = \underline{D}_{t-1} \cup \{\underline{x}_t\} \quad \Rightarrow \quad m_t = m_{t-1} + 1 \quad \Rightarrow \quad ii_t = [ii_{t-1}, t]$$

b. Calculamos la matriz inversa de kernel reducida de forma iterativa usando el lema de inversión:

$$\underline{\underline{\tilde{K}}}_t^{-1} = \frac{1}{\delta_t} \begin{bmatrix} \delta_t \underline{\underline{\tilde{K}}}_{t-1}^{-1} + \underline{a}_t \underline{a}_t^T & -\underline{a}_t \\ -\underline{a}_t^T & 1 \end{bmatrix}$$

[3] Siendo $m_{t-1}$ la longitud del diccionario para el instante t-1.

Opcionalmente se puede calcular la matriz del kernel reducida de forma iterativa, pero no es necesario su uso, ya que el algoritmo funciona en tiempo real y no es necesario ir guardando la matriz, solo su inversa.

$$\underline{\underline{\tilde{K}}}_t = \begin{bmatrix} \underline{\underline{\tilde{K}}}_{t-1} & \underline{\tilde{k}}_{t-1}(\underline{x}_t) \\ \underline{\tilde{k}}_{t-1}(\underline{x}_t)^T & k_{tt} \end{bmatrix}$$

c. Actualización de la matriz $\underline{\underline{P}}_t$: Se hace también de forma iterativa, ya que al introducir un nuevo elemento en el diccionario, los pesos asociados a esa nueva entrada son un vector del tipo $\underline{a}_t = [0 \cdots 01]$ que se añaden como una nueva fila a $\underline{\underline{A}}_t$. Para recuperarlo, solo depende de su entrada en el diccionario, por ser tratado como linealmente independiente:

$$\underline{\underline{P}}_t = \left(\underline{\underline{A}}_t^T \underline{\underline{A}}_t\right)^{-1} = \begin{bmatrix} \underline{\underline{P}}_{t-1} & 0 \\ 0 & 1 \end{bmatrix}$$

d. Actualización de los pesos sobre el kernel reducido, $\underline{\tilde{\alpha}}_t$:

$$\underline{\tilde{\alpha}}_t = \underline{\underline{\tilde{K}}}_t^{-1} \underline{\underline{P}}_t \underline{\underline{A}}_t^T y_t = \underline{\underline{\tilde{K}}}_t^{-1} \begin{bmatrix} \underline{\underline{P}}_{t-1} \underline{\underline{A}}_{t-1}^T y_{t-1} \\ y_t \end{bmatrix}$$

Definiendo $d_t = y_t - \underline{\tilde{k}}_{t-1}(\underline{x}_t)^T \underline{\tilde{\alpha}}_{t-1}$, como el error de la estimación en 't' de $y_t$ con el diccionario de $\underline{\underline{D}}_{t-1}$(es decir el error de dependencia lineal que se cometería de aproximar la $y_t$ con las entradas del diccionario del instante anterior), tenemos:

$$\underline{\tilde{\alpha}}_t = \begin{bmatrix} \underline{\tilde{\alpha}}_{t-1} - \dfrac{\underline{a}_t}{\delta_t} d_t \\ \dfrac{d_t}{\delta_t} \end{bmatrix}$$

e. Opcionalmente se podría calcular la matriz de los coeficientes de combinación lineal de las entradas con

respecto a los elementos del diccionario sobre el subespacio característico $\mathcal{H}$.

$$\underline{\underline{A}}_t = \begin{bmatrix} \underline{\underline{A}}_{t-1} & 0 \\ 0 & 1 \end{bmatrix}$$

Se plantea su cálculo de forma iterativa, pero no es necesario para la implementación del algoritmo ya que es online y en tiempo real, a no ser que se desee la recuperación de las estimaciones en bloque al finalizar el algoritmo.

- Si $\delta_t < \nu$, no se altera el diccionario ya que se considera que la nueva entrada se puede poner como combinación lineal de las entradas del diccionario anteriores a partir de unos pesos $a_t$ en el subespacio característico:

$$\underline{\underline{D}}_t = \underline{\underline{D}}_{t-1} \quad \Rightarrow \quad \underline{\underline{\tilde{K}}}_t = \underline{\underline{\tilde{K}}}_{t-1}$$

a. Definimos $\underline{q}_t$:

$$\underline{q}_t = \frac{\underline{\underline{P}}_{t-1}\underline{a}_t}{1 + \underline{a}_t^T \underline{\underline{P}}_{t-1}\underline{a}_t}$$

b. Actualizamos $\underline{\underline{P}}_t$:

$$\underline{\underline{P}}_t = \underline{\underline{P}}_{t-1} - \frac{\underline{\underline{P}}_{t-1}\underline{a}_t\underline{a}_t^T\underline{\underline{P}}_{t-1}}{1 + \underline{a}_t^T \underline{\underline{P}}_{t-1}\underline{a}_t}$$

c. Calculamos los pesos en el espacio origen, $\underline{\tilde{\alpha}}_t$:

$$\underline{\tilde{\alpha}}_t = \underline{\underline{\tilde{K}}}_t^{-1}\underline{\underline{P}}_t\underline{\underline{A}}_t^T y_t = \underline{\tilde{\alpha}}_{t-1} + \underline{\underline{\tilde{K}}}_t^{-1}\underline{q}_t d_t$$

Siendo $d_t = y_t - \underline{\tilde{k}}_{t-1}\left(\underline{x}_t\right)^T \underline{\tilde{\alpha}}_{t-1}$.

d. Podemos opcionalmente calcular la matriz $\underline{\underline{A}}_t$ sobre las mismas hipótesis que comentamos en el apartado e) de la otra opción.

$$\underline{\underline{A}}_t = \begin{bmatrix} \underline{\underline{A}}_{t-1} \\ \underline{a}_t \end{bmatrix}$$

#### 4.2.2.3 Paso 3: Repetir

Volver al Paso 2 hasta que se acaben las muestras de entrenamiento.

#### 4.2.2.4 Paso 4: Salida

La salida del algoritmo es $\underline{\underline{D}}_t$, $\underline{\tilde{\alpha}}_t$ y opcionalmente $\underline{\underline{A}}_t$ y $\underline{\underline{\tilde{K}}}_t$ para el caso de querer recuperar todas las estimaciones en forma de bloque al finalizar el algoritmo con la siguiente expresión:

$$\underline{\hat{y}} = \underline{\underline{A}}_t \underline{\underline{\tilde{K}}}_t \underline{\tilde{\alpha}}_t$$

De esta ecuación podemos distinguir dos tipos de coeficientes:

- $\underline{\underline{A}}_t$: Agrupa $\underline{a}$, que son los coeficientes de las combinaciones lineales de los vectores de entrada sobre los elementos del diccionario sobre el espacio característico (sobre todos los vectores se le aplica la función kernel para llevarlos a $\mathcal{H}$). En cierta medida lleva nuestra solución reducida a la solución completa, es decir $\underline{\underline{\tilde{K}}}_t \rightarrow \underline{\underline{K}}_t$.

- $\underline{\tilde{\alpha}}_t$: Son los pesos que nos permiten recuperar la información de $\underline{y}$ en función de los vectores de entrada en el espacio de partida. También se puede ver como el vector que marca la ponderación en $\mathcal{H}$ de la matriz de kernel reducida (linealmente indep.) que es necesaria para que al volver al espacio origen recuperemos $\underline{y}$.

En principio el algoritmo no esta pensado para la recuperación de las estimaciones en modo bloque, sino en modo online y en tiempo real, por lo que se

recupera cada estimación $\hat{y}_t$ en cada iteración del algoritmo, de la siguiente forma:

$$\hat{y}_t = \underline{\tilde{k}}_t \left( \underline{x}_t \right)^T \underline{\tilde{\alpha}}_t$$

## *4.3 Series de Volterra*

Para la implementación de un método que utilizara las series de Volterra, hemos utilizado la siguiente referencia [C.-H. Tseng, E.J. Powers; 1993] con título "Nonlinear Channel Equalization in Digital Satellite Systems". En él, se intenta dar un nuevo diseño de un igualador no lineal para un enlace satelital con modulación PSK.

Para realizar nuestro fin, se utiliza el teorema del punto fijo, junto con las propiedades de los mapas de contracción. De esta forma se recuperan los símbolos de entrada de una manera iterativa.

El objeto de estudio del documento de referencia, es la distorsión no lineal introducida por el efecto de saturación del amplificador del satélite en un sistema digital satelital. Este efecto limita de manera muy importante las prestaciones de nuestro enlace. Con objetivo de evitar este efecto, se hace operar al amplificador en una zona de unos pocos decibelios por debajo del límite de saturación, sin embargo, en esta situación perdemos potencia en las trasmisiones, lo cual tampoco es deseable en este tipo de enlaces, ya que las transmisiones demandan mucho ancho de banda, con lo que es necesaria un alto nivel de modulación, con lo que la eficiencia espectral debe ser alta.

No es de extrañar que ante la presencia de las no linealidades comentadas, el uso de los igualadores lineales tradicionales no sea adecuado, con lo que es necesario el empleo de técnicas no lineales.

La técnica que vamos a emplear utiliza una estructura de Volterra que modela nuestro canal no lineal. Además, tal y como hemos comentado, consideramos la igualación no lineal como un problema de punto fijo, que nos permite una aproximación sucesiva para eliminar progresivamente las no linealidades del canal. De esta forma, disponemos de un método flexible y potente, ya que las capacidades de éste, dependen del número de etapas de procesamiento que incluyamos en la cascada, con lo que no modificamos la estructura original.

### 4.3.1 Representación de un canal no lineal con series de Volterra

Un canal de comunicaciones paso banda no lineal puede ser modelado bajo una serie de Volterra compleja, tal y como se muestra en la expresión 4.18, donde $x_n$ son los símbolos transmitidos y $y_n$ los recibidos, ambos complejos. Como podemos ver en la expresión 4.18, la función $V(\cdot)$ depende de los instantes de entrada $x_n,\ldots,x_{n-N+1}$.

Los coeficientes $h_1(\cdot)$, $h_3(\cdot)$, etc., son los kernel de la serie de Volterra que modelan nuestro canal. Debemos notar que solo aparecen los kernel impares en el desarrollo de la serie de Volterra, cuya explicación es debida a la naturaleza paso banda del canal no lineal.

$$y_n = \sum_{i=0}^{N-1} h_1(i)\, x_{n-i} + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} h_3(i,j,k)\, x_{n-i}x_{n-j}x_{n-k}^{*} + .... \qquad \textbf{4.18}$$

$$y_n = V\left(x_n, x_{n-1},\ldots,x_{n-N+1}\right)$$

En nuestro ejemplo en particular, usamos una constelación PSK, que es uno de los esquemas de modulación más popular, usado principalmente en comunicaciones digitales satelitales. La expresión de una constelación M-PSK[4] puede verse en la expresión 4.19, donde A corresponde con la amplitud constante para todos nuestros símbolos, y $\phi_k$ con la fase.

$$\begin{aligned} s_k &= Ae^{j\phi_k} \\ \phi_k &= \frac{2\pi\left(k-1/2\right)}{M} \end{aligned} \qquad \textbf{4.19}$$

La elección de la modulación PSK como ejemplo no ha sido realizada al azar, ya que es una constelación, que al estar basada en cambios de fase, y no de amplitud, es muy resistente a las no linealidades. Por esta razón, podemos truncar nuestra serie de Volterra en términos relativamente bajos, ya que no necesitamos tener una estimación de nuestro canal demasiado exhaustiva.

[4] Tenemos M símbolos: $s_k : k = 1,\ldots,M$ .

### 4.3.2 Problema de punto fijo

Nuestro objetivo es el de resolver nuestra igualación no lineal usando el algoritmo del punto fijo. Antes de ponernos en materia con el problema de igualación, debemos recordar la expresión del punto fijo que se puede ver en 4.20. Un punto $x_f$ que satisfaga la expresión 4.20 es denominado un punto fijo, o lo que es lo mismo, que es invariante ante la transformación $T(\cdot)$. Un punto fijo no tiene porque ser único, y puede no existir.

$$x = T(x) \tag{4.20}$$

Para encontrar nuestro punto fijo, usamos un método clásico que está basado en la utilización del teorema de contracción. Este teorema nos permite encontrar nuestro punto fijo si la transformación $T(\cdot)$ es de contracción, y viene a decir que $T(\cdot)$ es una transformación de contracción si existe un $\alpha$ con $0 \le \alpha < 1$ que hace cumplir la expresión 4.21 para todo $x_1, x_2$ perteneciente al subespacio.

$$\left\| T(x_1) - T(x_2) \right\| \le \alpha \left\| x_1 - x_2 \right\| \tag{4.21}$$

Este teorema lo que nos intenta decir es que para cualquiera dos puntos, la distancia entre ellos después de la transformación no es mayor que la distancia entre ellos antes de ésta.

Después de lo dicho, podemos extraer que si $T(\cdot)$ es una transformación de contracción, sobre un subespacio S de un espacio de Banach, tenemos que existe un único punto $x_f$ que satisface la expresión 4.20. Además tenemos que si partimos de un punto arbitrario, $x_0$, podemos obtener $x_f = \lim_{n\to\infty} T^n(x_0)$, o lo que es lo mismo, realizando la transformación de forma recursiva, lo cual nos da un método para encontrar nuestro punto fijo. Por último, también tenemos que añadir como corolario del teorema, que bajo las mismas hipótesis mencionadas anteriormente, y teniendo $x_{n+1} = T(x_n)$ se cumple la desigualdad expresada en 4.22, la cual nos ofrece un umbral de la rapidez con la que podemos encontrar

nuestro punto fijo, basándonos en el parámetro $\alpha$ de la transformación de contracción.

$$\left\| x_n - x_f \right\| \leq \frac{\alpha^n}{1-\alpha} \left\| x_1 - x_0 \right\| \quad \textbf{4.22}$$

### 4.3.3 Igualación no lineal

Una vez contado todo lo anterior, estamos en disposición de tomar todo y enfocarlo a nuestro problema de igualación no lineal de canal. Disponemos de unos símbolos de salida del canal a la entrada de nuestro receptor a los cuales denominamos $y_k$. Por otro lado tenemos que identificar los símbolos transmitidos, que denotamos por $x_k$, utilizando $y_k$ así como los símbolos $x_k$ transmitidos anteriormente, que se suponen conocidos.

Utilizando la expresión 4.18, tenemos que $y_k = V(x)$. Apoyándonos en un problema de punto fijo podemos llegar a la expresión 4.23 empleando 4.20, donde $\beta$ es una constante.

$$T(x) = x + \beta\left[ y_k - V(x) \right] \quad \textbf{4.23}$$

Podemos notar que resolver la expresión 4.23 equivale a encontrar el punto fijo tal que $x = x_k$, con lo que particularizando, podemos reescribir la ecuación 4.23 y obtener la 4.24

$$T(x_k) = x_k + \beta\left[ y_k - V(x_k) \right] = x_k \quad \textbf{4.24}$$

Hemos de recordar que nos encontramos bajo la hipótesis del teorema del punto fijo, comentado en el punto 4.3.2, con lo que suponemos que nuestra $T$ es una trasformación de contracción[5]. Esto implica la unicidad y existencia del punto

[5] La condición para que T produzca un mapeo de contracción, está relacionada directamente con la elección del parámetro $\beta$.

fijo, así como la forma de obtenerlo mediante $x_f = \lim_{n\to\infty} T^n(x_0)$ y $x_{n+1} = T(x_n)$. Por lo tanto, podemos utilizar el teorema del punto fijo para realizar nuestra igualación no lineal de canal.

Examinando las expresiones que emplearemos para obtener nuestro punto fijo, debemos notar que evoca hacia una estructura en la que entren en juego combinaciones de múltiples unidades de procesamiento. Cada unidad de procesamiento realiza la misma transformación $T(\cdot)$. Suponiendo que estamos en la i-esima unidad, deberemos realizar la trasformación que muestra la expresión 4.25. De acuerdo con el teorema del punto fijo, la secuencia $\{z_n\}$ converge a nuestro punto fijo, $x_k$.

$$z_{i+1} = z_i + \beta\left[y_k - V(z_i)\right] \qquad \textbf{4.25}$$

Si volcamos nuestra expresión 4.25 en una estructura recursiva implementable de manera practica, podemos llegar a la Figura 4.1 que nos muestra el ejemplo de estructura sobre la que resolveríamos nuestro problema de punto fijo, y en consecuencia nuestra igualación no lineal de canal.

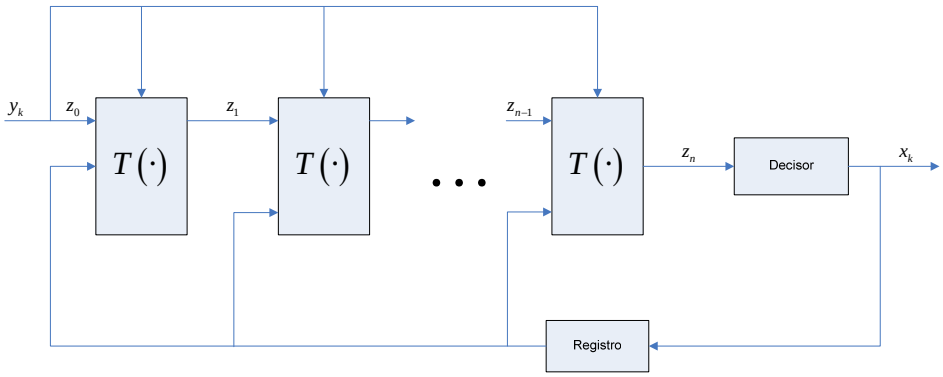



**Figura 4.1 – Esquema iterativo de cálculo de punto fijo.**

Con el fin de obtener una cota de funcionamiento de nuestro sistema, podemos proponer un límite de velocidad de convergencia de nuestro sistema ante un número idéntico de etapas de procesamiento[6]. Este límite esta muy directamente relacionado con la expresión 4.22 mostrada en el teorema del punto

[6] Entendemos que a mayor número de etapas de procesamiento, tendremos mejor estimación de nuestro punto fijo.

fijo. En ella podemos ver que el parámetro $\alpha$ controla la tasa de convergencia de la secuencia $\{z_n\}$. Se puede ver que para obtener una convergencia rápida, deberemos tener una $\alpha$ lo mas pequeña posible. Si suponemos que $\alpha$ toma el mínimo valor posible, es decir $\alpha = 0$, podemos llegar a una relación que limita la convergencia de nuestro algoritmo, relacionándola con nuestra constante $\beta$, tal y como se puede ver en la expresión 4.26, donde $h_1(0)$ corresponde con el término de orden 1 asociado al retardo $\tau = 0$ de nuestro modelo de canal en serie de Volterra.

$$\beta = \frac{1}{h_1(0)} \qquad \textbf{4.26}$$

## *4.4 Clasificadores*

En este apartado vamos a detallar la solución de nuestro problema de igualación de canal no lineal, orientándola desde el punto de vista de algoritmos clasificadores. Para nuestros fines utilizaremos un algoritmo clasificador muy conocido y denominado K Nearest Neighbours (KNN)[7], y nos basaremos en un documento de referencia que emplea este tipo de métodos en el marco en el que nos encontramos. Dicho documento es [P.Savazzi, L. Favalli, E. Costamagna, A. Mecocci; 1998] y tiene como título "A Suboptimal Approach to Channel Equalization Based on the Nearest Neighbour Rule".

Este documento ilustra una implementación de un sistema de comunicaciones móviles GSM, en el que examinamos el uso de un receptor que es usado para la igualación de canal, así como la demodulación, mediante el algoritmo clasificador KNN. El método que proponen, realiza un procesamiento símbolo a símbolo, suponiendo que el conocimiento del canal se encuentra en el proceso de mapeo entre los símbolos recibidos y los símbolos de la secuencia de entrenamiento.

La mayor ventaja que obtenemos al utilizar este tipo de métodos no óptimos en detección, es la de reducir la complejidad computacional en comparación con el método MLSE[8], que en este caso es el detector óptimo.

La elección del canal de comunicaciones móvil radio, no ha sido realizada al azar, sino que viene fundamentada en que es un canal muy complejo y variante, ya que disponemos de atenuación multitrayecto, así como dispersión en tiempo y frecuencia Doppler. Estos efectos, nos producen consecuencias no deseables en nuestras trasmisiones, como interferencia intersimbólica (ISI), o variaciones en tiempo de la respuesta al impulso de nuestro canal (CIR). Para contrarrestar estos efectos nocivos, la arquitectura GSM usa un receptor MLSE basado en el algoritmo de Viterbi, ya comentado con anterioridad, el cual nos ofrece la solución óptima de detección de información ante presencia de ISI y ruido aditivo gaussiano.

[7] Del inglés "K Nearest Neighbors", que viene a significar "K Vecinos Próximos".
[8] Del inglés "Maximum Likelihood Sequence Estimation" que significa "Estimador de Secuencia de Máxima Verosimilitud".

No obstante la asunción de que el algoritmo de Viterbi nos ofrece la solución óptima en este tipo de entornos está sujeta al conocimiento del comportamiento estadístico del canal, y tal y como hemos comentado, eso no es del todo cierto en nuestro entorno de comunicaciones particular, ya que nuestro canal es rápidamente variante. La solución que se ofrece es la estimación adaptativa de canal, y se realiza insertando secuencias conocidas de entrenamiento en la información a enviar. Examinando estas cadenas conocidas en nuestro receptor, seremos capaces de recalcular las métricas para la trama de información que estamos enviando[9].

Podemos decir que el problema de igualación de canal puede ser entendido como un problema de filtrado inverso, y el problema de demodulación como un mapeo entre unas señales recibidas y unos símbolos esperados. Por esta razón, el objetivo es unificar ambos problemas (igualación y demodulación) y tratarlos como un problema de clasificación.

Desde el punto de vista del documento abordan el problema de clasificación desde un principio simplista, evitando algoritmos neuronales más complejos, y volviendo a las soluciones iniciales de los problemas de clasificación. Este tipo de soluciones, son simples, pero a su vez robustas, y tienen como respaldo que se han aplicado en multitud de problemas. En particular utilizan el algoritmo de los K vecinos más cercanos (KNN).

Para comprobar las prestaciones del algoritmo KNN en el problema de igualación/demodulación, lo aplican a un caso práctico, en el ámbito de un sistema de radio comunicaciones móviles. El receptor realiza su decisión símbolo a símbolo, comparando el valor de cada bit, con aquellos bits de la secuencia de entrenamiento, asignando el valor cero o en su caso el valor del bit mas parecido. Esta técnica es subóptima en comparación con los igualadores MLSE en los cuales las decisiones están referidas a la secuencia de datos completa, aprovechando de esta manera la interdependencia de los símbolos. A pesar de ser una aproximación subóptima, tenemos la contraprestación de que la complejidad es mucho menor, con lo que aunque no igualemos las prestaciones del algoritmo de Viterbi en este tipo de escenarios, bien es cierto que el algoritmo

[9] Hemos de notar que esta solución implica que la complejidad del sistema sea exponencial con la memoria del sistema, al estar directamente relacionado con el número de estados usados en el diagrama de Trellis. Como posible solución tenemos el uso de algoritmos ‘capados’ que reduzcan el espacio de estados de nuestro Trellis.

si podrá llegar al nivel de algoritmos simplificados, con un menor grado de complejidad que estos[10].

### 4.4.1 Clasificación de patrones

El problema de clasificación de patrones no es más que interpretar señales corruptas y mapearlas en unas clases teóricas, que en el caso de la demodulación corresponden con la constelación de la modulación usada. Está técnica sabemos que es muy empleada en el tipo de problemas que nos esperan de modulación-igualación.

En particular vamos a centrarnos en el clasificador en el que basamos nuestro trabajo, el clasificador NN.

Bajo la hipótesis del pleno conocimiento de las probabilidades del problema, sabemos que la teoría de decisión de Bayes nos ofrece la tasa de error óptima. No obstante, cuando la información de probabilidades no está presente, debemos obtenerla de las muestras observadas. Este tipo de algoritmos son los denominados no paramétricos, tal y como detallamos en la introducción del presente proyecto fin de carrera. En particular emplea la distancia entre muestras, o cualquier otro tipo de medidas de similitud como manera de clasificación.

Bajo este marco de funcionamiento, nos encontramos el algoritmo NN, y como extensión el algoritmo K-NN, que será el que empleemos en nuestras simulaciones.

### 4.4.2 Descripción del algoritmo

Suponiendo que $y$ es la muestra observada en el instante actual, y siendo $X_t = \{x_1, x_2, \ldots, x_n\}$ un conjunto de n muestras etiquetadas[11]. Si tenemos que

[10] Respetando en cualquier caso los límites que impone las especificaciones de GSM, a fin de cumplir el ejemplo práctico planteado.
[11] Suelen ser los patrones de entrada al sistema, en los cuales vamos a clasificar nuestras salidas.

$x_i \in X_t$ es la muestra más cercana[12] a $y$, entonces de acuerdo con el algoritmo NN, clasificaremos a $y$ con la etiqueta asociada a $x_i$. Este concepto tan simple, da lugar a un algoritmo muy rápido y eficiente desde el punto de vista de implementación HW.

Tal y como hemos comentado, el algoritmo NN es un caso particular del algoritmo más general K-NN. En este caso, la etiqueta asociada con la muestra $y$ es la mas frecuente entre aquellas asignadas a la 'K' muestras etiquetadas mas similares, con el mismo criterio que teníamos en el caso del algoritmo NN. Para K>1 debemos tener cuidado para evitar los posibles problemas entre clases, es decir, podemos encontrarnos con el caso de que la suma de todas las distancias entre la muestra actual y el resto de muestras asignadas a cada clase tengan la misma densidad, por lo que tendremos un empate entre las clases competidoras por etiquetar la muestra actual $y$.

En el caso de encontrar problemas de equipoblación entre las clases competidoras, se propone como método de desempate entre estas, la suma de todas las distancias de los K vecinos de las clases afectadas. La menor distancia de ese conjunto, será finalmente la etiqueta otorgada a nuestra muestra actual, $y$.

Tal y como hemos podido notar en la explicación del algoritmo NN, es un procedimiento subóptimo, porque nos lleva a una tasa de error mayor que el posible mínimo que alcanzaríamos con métodos Bayesianos. No obstante, se ha encontrado de manera teórica [R. O. Duda, P.E. Hart; 1973] que con un número ilimitado de muestras, tenemos que la tasa de error con el algoritmo NN no es peor en cualquier caso que dos veces la tasa de error de Bayes.

### 4.4.3 Análisis de complejidad del algoritmo

Al inicio del punto 4.4 comentamos que el algoritmo del documento de referencia en el cual nos íbamos a basar para construir nuestra simulación, aplicaba el algoritmo NN de nuestro interés para un caso práctico de igualación no

[12] Depende de la métrica de distancia que elijamos.

lineal de canal sobre la arquitectura GSM con la finalidad de resolver de manera conjunta el problema de igualación y demodulación. Como es evidente nuestro objetivo en el presente proyecto, no es verificar que el algoritmo KNN es funcionalmente correcto en el ámbito de comunicaciones móviles GSM, sino utilizar este algoritmo clasificador en el mismo marco que los autores lo hacen, es decir, igualación no lineal de canal. Nuestro objetivo es únicamente reforzar las conclusiones que obtengamos para nuestro algoritmo principal KRLS. Por esta razón, y a pesar de haber comentado el problema práctico sobre el que aplican el algoritmo KNN los autores del documento, no iremos mas allá en ese problema, ni comentando la arquitectura GSM, ni los resultados prácticos obtenidos en ese campo, ya que carece de sentido para lo que nos ocupa. A partir de esta medida, solo comentaremos los aspectos genéricos del algoritmo propuesto por los autores, ya que será lo que emplearemos nosotros en nuestras simulaciones de igualación no lineal.

Lo más interesante del empleo del algoritmo KNN en este tipo de problemas, es que la complejidad es mucho menor que la que nos ofrece el algoritmo de Viterbi. Además no es necesario obtener la respuesta al impulso del canal inicialmente, con lo que se reduce de forma significativa la carga computacional. Esto es debido a que realizamos un mapeo entre los patrones de entrada al sistema que nos sirven como etiquetas y nuestras salidas del sistema que será lo que etiquetaremos. Ese mapeo propuesto no entiende del canal, ya que lo único que realiza son medidas de distancia sobre el espacio de salida de nuestro sistema. Por esta razón es muy importante disponer del algoritmo correctamente inicializado, ya que al tener una base de salidas correctamente etiquetadas, podemos calcular de forma más efectiva las etiquetas de las siguientes salidas $y$ en los instantes posteriores, obviando los errores en la inicialización y tomando únicamente los derivados de la aplicación del algoritmo en cuestión.

## *4.5 Redes Neuronales*

Para presentar el uso de redes neuronales en la solución de problemas de igualación no lineal de canal, hemos elegido un documento de referencia [M. Nicolae, C. Botoca, G. Budura; 2004] con título “Nonlinear Complex Channel Equalization Using A Radial Basis Function Neural Network”, con el que pretendemos dar el contrapunto a nuestro algoritmo kernelizado KRLS desde el campo de las redes neuronales.

Para detallar el algoritmo vamos a centrarnos en su documento de referencia y comenzaremos englobándolo en el entorno del problema general.

En la actualidad, en las redes de comunicaciones de alta velocidad existe un grave problema, la presencia de interferencia intersimbólica (ISI). Dicha interferencia es producida principalmente por dispositivos pasivos y activos que introducen distorsiones no lineales que afectan a las señales.

El problema de igualación puede ser tratado como un problema de clasificación de señales. Por esta razón las redes neuronales son viables para su uso en este tipo de problemas, ya que son capaces de crear regiones de decisión complejas arbitrariamente. Apoyándonos en estudios, podemos decir que los igualadores neuronales son superiores en rendimiento, en comparación con los igualadores tradicionales en condiciones de altas distorsiones no lineales y señales muy variantes.

Actualmente se han implementado múltiples arquitecturas de igualadores neuronales, muchos de ellos basados en montajes LTE[13]. Este tipo de estructuras, permiten que el montaje LTE elimine las distorsiones lineales y la ISI, y que la red neuronal compense las no linealidades.

Los métodos basados en redes neuronales de perceptrón multicapa (MLP) necesitan largas cadenas de entrenamiento, y no nos aseguran que encontremos el mínimo global de la función. En comparación, las redes RBF, nos dan una

[13] Del inglés “Linear Transversal Filter” que viene a significar Filtro Transversal Lineal.

solución rápida y robusta. Además, las redes neuronales RBF, tienen una estructura similar al igualador óptimo Bayesiano (con decisión símbolo a símbolo), el cual no nos garantiza encontrar un mínimo de error cuadrático medio (MSE), pero si la solución con minima tasa de error binaria media (BER). Por esta razón, podemos decir que las redes RBF son un candidato ideal para implementar el igualador óptimo Bayesiano, teniendo mejores prestaciones que los igualadores basados en LTE y MLP, con una estructura más simple y un entrenamiento más eficiente.

Centrándonos en las redes neuronales RBF para la construcción de nuestro igualador no lineal, tendremos como siguiente paso obtener el algoritmo de aprendizaje que utilizaremos para actualizar los parámetros de nuestra red RBF. El algoritmo más popular consiste en reglas de aprendizaje no supervisado para los centroides de las neuronas ocultas, y reglas de aprendizaje supervisado para los pesos de las neuronas de salida. Para la actualización de los centros normalmente se utiliza el algoritmo k-medias de agrupamientos, que consiste en calcular la distancia (normalmente se elige la distancia de orden cuadrático) entre el vector de entrada y los centros, eligiendo la menor, y moviendo el centro correspondiente en la dirección del nuevo vector de entrada. Sin embargo este método de actualización de centros tiene ciertos problemas, como por ejemplo, que la clasificación depende de los valores iniciales que tienen los centros de la red RBF, de la definición de distancia escogida, del número de clases del agrupamiento, etc. Esto conlleva que si un centro es elegido de forma inapropiada, puede no ser actualizado nunca, con lo que no representaría una clase del agrupamiento.

Para solucionar estos contratiempos, los autores del documento de referencia [M. Nicolae, C. Botoca, G. Budura; 2004] proponen un nuevo método competitivo para actualizar los centros de nuestra red neuronal RBF, el cual recompensa a la neurona ganadora y penaliza a la segunda ganadora, a la cual denominan “rival”. El algoritmo propuesto es muy simple y con unas altas prestaciones. Crea las clases automáticamente a la salida de la red, y dispone de centros y pesos sinápticos complejos, aunque la no linealidad de las neuronas ocultas la forman funciones reales. Empleando todo esto, tenemos que nuestro igualador competitivo RBF es capaz de aproximar una función no lineal arbitraria en un espacio multidimensional con una reducida complejidad computacional en comparación con otros algoritmos.

### 4.5.1 Problema de igualación

El marco sobre el que va a funcionar nuestra red RBF es el que hemos comentado con anterioridad en el capítulo destinado a la introducción del presente Proyecto Fin de Carrera. No obstante, vamos a tratar de explicar la versión que tienen del problema los autores del documento de referencia [M. Nicolae, C. Botoca, G. Budura; 2004] a fin de unificar ideas entre los distintos métodos que estamos comentando en el presente capítulo.

El problema de igualación es visto históricamente desde el punto de vista de un problema de filtro inverso. Un esquema del problema puede ser visto en la Figura 4.2. En este sentido, los igualadores se diseñan para seguir las variaciones temporales de los canales, ajustando sus coeficientes para mantener una determinada SNR[14]. La presencia de ruido hace que normalmente estos igualadores sean subóptimos. Para sortear este tipo de problemas, se puede usar otro punto de vista alternativo, al considerar el problema de igualación como un problema de clasificación de patrones.

A través de este nuevo camino, tenemos que nuestro nuevo objetivo de igualación pasa por separar nuestros símbolos recibidos en el espacio de salida, el cual tendrá generalmente fronteras de decisión no lineales. En este punto es donde introducimos las redes neuronales, ya que son muy conocidas por su habilidad para resolver el problema de clasificación utilizando fronteras muy complejas. En esta aplicación las redes neuronales muestran que tienen un gran poder en señales con no linealidades altas y muy variantes temporalmente.

[14] Del inglés “Signal Noise Ratio” que significa “Relación Señal a Ruido”.

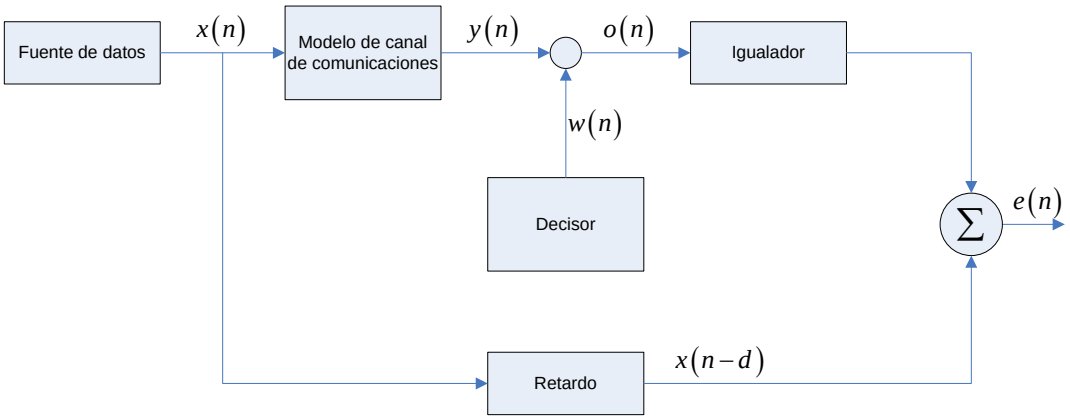



**Figura 4.2 – Esquema de igualación de canal**

Revisando la Figura 4.2, podemos determinar de una manera más formal el comportamiento que debe tener nuestro clasificador basado en red neuronal. Suponiendo que $\hat{x}(n)$ es la estimación del patrón de entrada, usando la señal $o(n)$ y la señal deseada retardada $x(n-d)$, podemos extraer que ya que nuestro igualador tiene que clasificar la señal recibida en una de las N posibles clases, tendremos que deberá cumplirse la expresión 4.27, que de manera más específica viene a significar la expresión 4.28.

$$P_{m,d} = \bigcup_{1 \le l \le N} P_{m,d}(l) \qquad \textbf{4.27}$$

$$P_{m,d}(l) = \left\{ y(n) \mid x(n-d) = x^{(l)} \right\} \quad \forall l : 1 \le l \le N \qquad \textbf{4.28}$$

## 4.5.2 Modelo de canal de comunicaciones

El modelo que vamos a emplear para modelar nuestro canal de comunicaciones es un modelo no lineal, tal como se puede observar en la Figura 4.3. En ella vemos que para modelar nuestro canal, tenemos dos bloques, uno de ellos que introduce el comportamiento lineal, y el otro las distorsiones no lineales. La parte lineal la podemos expresar con un filtro transversal FIR que puede definirse como aparece en la expresión 4.29, donde $a_i$ corresponde con los coeficientes del filtro y 'k' con el orden del filtro.

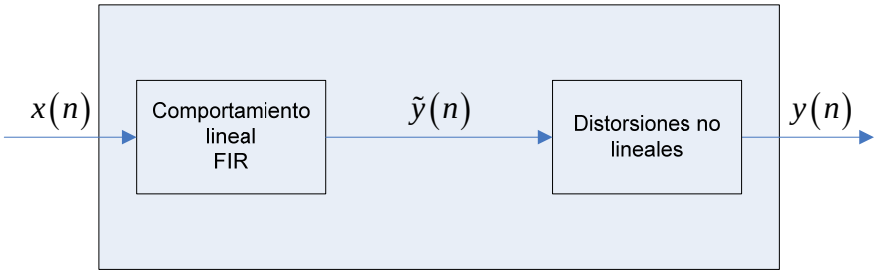



**Figura 4.3 – Canal de comunicaciones no lineal.**

$$\tilde{y}(n) = \sum_{i=0}^{k-1} a_i x(n-i) \qquad \textbf{4.29}$$

Por otro lado, la parte no lineal del modelo del canal de comunicaciones es muy importante, y podría corresponder con la expresión 4.30 de manera general.

$$y(n) = \sum_{i=1}^{k-1} a_i \tilde{y}(n)^i \qquad \textbf{4.30}$$

Tenemos que notar también que a la salida de nuestro modelo de canal, $y(n)$, tendremos que añadir el ruido gaussiano $w(n)$ con media 0 y varianza $\sigma^2$, con lo que obtendremos la salida final del sistema, que se puede ver en la expresión 4.31

$$o(n) = y(n) + w(n) \qquad \textbf{4.31}$$

### 4.5.3 Igualador RBF complejo

Tal y como hemos dicho anteriormente, podemos construir la parte lineal de nuestro igualador mediante un filtro FIR transversal. Para conseguir las distorsiones no lineales, colocaremos a la salida de nuestro filtro FIR nuestra red neuronal, tal y como se puede ver en la Figura 4.3.

Para nuestro filtro lineal, suponiendo que 'm' es el orden del filtro FIR, tenemos que su salida es $o = \left[ o(n)\ o(n-1)\ \ldots\ o(n-m+1) \right]$ , lo cual será la entrada de la red neuronal RBF.

La arquitectura de la parte no lineal de nuestro modelo se puede ver en la Figura 4.4, y se basa simplemente en una red neuronal RBF. La capa oculta está formada por un array de neuronas de procesado, cada una de ellas con un parámetro $c_i$, que es su centro. Cada neurona calcula una distancia entre su centro y el vector de entrada de la red. La distancia puede ser definida de múltiples formas, dejando este aspecto abierto a la creatividad del implementador. Debemos dividir nuestra distancia por un parámetro $\rho_i$, el cual es la varianza del centro de la neurona de la capa oculta sobre la cual estemos trabajando. El resultado se pasa a través una función de activación real y no lineal, $\phi_i(\cdot, \rho_i)$, que definimos en la expresión 4.32, donde 'o' es el vector de entrada complejo de dimensión $n_h$, $c_i$ es el vector centro de las funciones RBF, también de dimensión $n_h$, $\rho_i$ es la varianza del centro y $n_h$ es el número de nodos de procesamiento.

$$\phi_i = \left[ \phi(o - c_i)^H (o - c_i), \rho_i \right] \quad 1 \le i \le n_h \qquad \textbf{4.32}$$

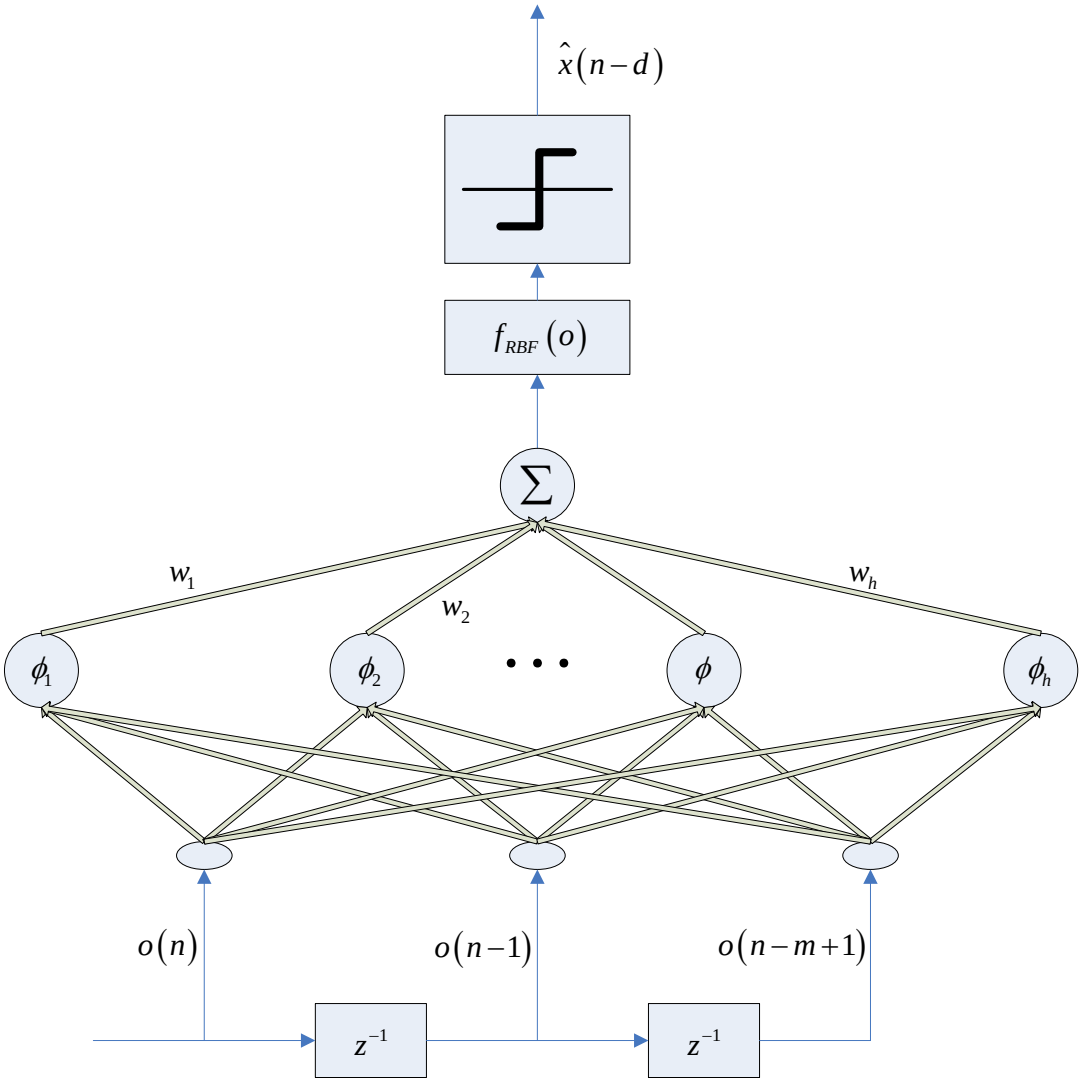



**Figura 4.4 – Arquitectura del igualador de red neuronal RBF.**

La función de salida no lineal suele ser una función gaussiana como se puede ver en la expresión 4.33, aunque en principio podría ser cualquier tipo. La elección de la gaussiana, está relacionada con el truco del kernel que se emplea en las redes neuronales RBF para hacer el mapeo de los vectores de entrada. Para ello empleamos una función infinita, que en este caso es la gaussiana.

$$\phi(\chi^2, \rho) = e^{\frac{\chi^2}{\rho}} \qquad \textbf{4.33}$$

El número de neuronas ocultas, $n_h$ está relacionado con el número de estados posibles del canal de salida $n_c$. En este apartado tenemos un

compromiso, ya que un $n_h > n_c$ genera un gasto de computación inútil, mientras que el caso de $n_h < n_c$ puede degradar las prestaciones de la red.

Con respecto a la varianza de los centros, $\rho$, y aprovechando la similitud con el igualador Bayesiano, lo definimos $\rho = 2 \cdot \sigma^2$, donde $\sigma^2$ es el ruido de dispersión que se define en la expresión 4.34, como la media del segundo momento de la diferencia entre el vector complejo de entrada y los centros de las neuronas de la capa oculta.

$$\sigma^2 = E\left\| o(n) - c_i \right\|^2 \qquad \textbf{4.34}$$

Con respecto a la capa de salida de la red, tendremos tantas neuronas como clases dispongamos en nuestro problema de clasificación[15]. En forma general, tendremos la función lineal a la salida que se define en la expresión 4.35, donde $w_i$ son los pesos complejos. Unificando la expresión 4.35 y la 4.33 podemos llegar a una relación más general que puede ser vista en la expresión 4.36.

$$f_{RBF}(o) = \sum_{i=1}^{n_h} \phi_i w_i \qquad \textbf{4.35}$$

$$f_{RBF}(o) = \sum_{i=1}^{n_h} w_i \cdot e^{\frac{(o-c_i)^H (o-c_i)}{\rho_i}} \qquad \textbf{4.36}$$

### 4.5.4 Aprendizaje competitivo penalizando al rival

El algoritmo propuesto de aprendizaje calcula la distancia entre el vector de entrada y los vectores centro de la red RBF. En este caso se propone como definición de distancia, la norma euclídea, tal y como aparece en la expresión 4.37.

$$\left\| o(n) - c_i(n) \right\| = \sqrt{\left| o(n) - c_i(n) \right|^2 + \ldots + \left| o(n-m+1) - c_i(n-m+1) \right|^2} \qquad \textbf{4.37}$$

[15] Si son clases complejas, deberemos tener una neurona para la parte real y otra para la parte compleja.

Suponiendo que la neurona 'j' es la que posee el centro con menor distancia al vector de entrada (expresión 4.38), determinamos que es la ganadora, con lo que movemos su centro $\eta$ veces hacia el vector de entrada.

$$j = \arg\min \left\| o(n) - c_i(n) \right\| \quad i = 1, \ldots, n_h \tag{4.38}$$

Hemos premiado a nuestra neurona ganadora, 'j', y nuestro algoritmo competitivo determina que tenemos que penalizar a la neurona rival, 'r', entendida como la segunda neurona ganadora para el mismo vector de entrada, la cual definimos en la expresión 4.39. En este caso, el centro del rival, se alejará $\gamma$ veces del vector entrada.

$$r = \arg\min \Big|_{i \neq j} \left\| o(n) - c_i(n) \right\| \quad i = 1, \ldots, n_h \tag{4.39}$$

El resto de neuronas no modifican el valor de sus centros, o lo que es lo mismo, en cada iteración sólo alteramos la neurona ganadora y su rival. En modo de resumen, el algoritmo de aprendizaje aparece en la expresión 4.40, donde $\eta$ y $\gamma$ son constantes de aprendizaje reales con valores comprendidos entre 0 y 1.

$$c_i(n+1) = \begin{cases} c_i(n) + \eta \left[ o(n) - c_i(n) \right] & , \ i = j \\ c_i(n) + \gamma \left[ o(n) - c_i(n) \right] & , \ i = r \\ c_i(n) & , \ i \neq j \neq r \end{cases} \tag{4.40}$$

Los parámetros $\eta$ y $\gamma$ se entienden como la velocidad de convergencia del algoritmo. Si elegimos $\eta$ mucho mayor que $\gamma$, la red RBF encontrará automáticamente el número de clases de salida, siempre y cuando $n_h$ sea mayor que el número de clases existentes. Los centroides RBF convergerán hacia los centros de las clases de las señales de entrada. Si por el contrario, tenemos que $n_h$ es menor que el número de clases, entonces la red RBF oscilará durante el entrenamiento, lo cual será síntoma de que el número de neuronas ocultas deberá ser incrementado.

### 4.5.5 Actualización de pesos

Hemos estado comentando como tratamos la capa oculta de nuestra red RBF, pero nos queda por comentar como tenemos en cuenta nuestra etapa de salida. Un algoritmo supervisado es usado para actualizar los pesos de estas neuronas. En particular, los autores del documento de referencia de este algoritmo, eligieron un LMS, el cual se detalla en la expresión 4.41, donde $\alpha$ está comprendida entre 0 y 1 y es la constante de aprendizaje.

$$w_i(n+1) = w_i(n) + \alpha \cdot e(n) \cdot \phi(n) \qquad \textbf{4.41}$$

Tal y como hemos comentado en los capítulos precedentes, el algoritmo LMS minimiza el error cuadrático medio. Aplicando esto a N secuencias de entrada, y suponiendo que $e_i(n)$ es el error complejo de la secuencia de entrada i-esima entendido en la forma de la expresión 4.42, podemos decir que el error cuadrático medio se corresponde con la expresión 4.43.

$$e(n) = x(n-d) - f_{RBF}(o) \qquad \textbf{4.42}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} e_i(n)^2 \qquad \textbf{4.43}$$

# Capítulo 5 Resultados Experimentales

## *5.1 Introducción*

En este quinto capítulo tenemos como objetivo explicar inicialmente como hemos implementado cada algoritmo comentado en el Capítulo 4. Posteriormente daremos resultados de su correcto o erróneo funcionamiento por separado, centrándonos en nuestro algoritmo bajo estudio KRLS, para finalizar el capítulo realizando una retrospectiva que nos permita relacionar nuestro algoritmo principal KRLS con el resto de algoritmos de referencia, cumpliéndose de esta manera el objetivo de nuestro proyecto fin de carrera.

Entrando en detalle en la organización de este capítulo, en la comprobación del funcionamiento de cada algoritmo, daremos un significado a cada uno de los parámetros empleados en su implementación, justificando su uso, siempre apoyados en su definición comentada en el Capítulo 4.

Finalmente tendremos que llegar al punto mas importante de todo el documento que nos ocupa. Este no será otro que el de comprobar el funcionamiento del algoritmo KRLS en comparación con otros algoritmos usados actualmente sobre un mismo problema y un mismo escenario, la igualación no lineal de canal. Este apartado lo dividiremos en tres experimentos que examinaran al KRLS frente a los algoritmos contrincantes. El primero de ellos lo haremos sobre un canal estático y para una sola estimación de test tras finalizar el entrenamiento. En el segundo, introduciremos en el esquema de aprendizaje no lineal una variación en los coeficientes del canal, con el objetivo de comprobar que prestaciones nos ofrecen los algoritmos ante un canal dinámico. Por último,

tendremos el experimento final del presente proyecto fin de carrera, y que consistirá en la modificación del esquema de entrenamiento dinámico y de los algoritmos para que funcionen guiados por decisión, o lo que es lo mismo, que los símbolos de entrenamiento que reciban, sean los mismos que ellos previamente estimen. Este último experimento, nos ayudará a comprobar como se expanden los errores en los algoritmos planteados, y si estos son estables. De esta forma conseguiremos evaluar las prestaciones de nuestros algoritmos ante un sistema variante, así como el tiempo que tarda cada uno en engancharse a la nueva situación, en caso de poder hacerlo. Tanto para el segundo como para el tercer experimento aprovecharemos los resultados del entrenamiento estático, ya que antes de introducir cualquier tipo de variación en el esquema, realizaremos un entrenamiento estático que nos garantice un punto de partida conocido.

Un detalle a tener muy en cuenta es que como es natural, no prestaremos el mismo nivel de profundidad a los algoritmos de referencia comentados en el Capítulo 4, que al algoritmo sobre el cual basamos todo nuestro estudio, (epígrafe 4.2). De esta forma comentaremos las prestaciones por separado de KRLS y del resto de algoritmos nos centraremos únicamente en las peculiaridades encontradas en las simulaciones realizadas, ya que el entorno de trabajo ya habrá sido detallado.

Continuando en este sentido, dejaremos de lado las muestras de las implementaciones de base de los algoritmos auxiliares desarrollados: KNN, RBF y Volterra, ya que en cualquier caso nos desembocarían a desviar nuestra atención del verdadero motivo del presente proyecto, que no es mas que demostrar el correcto funcionamiento del algoritmo KRLS y comparar sus prestaciones frente a otros algoritmos de igualación de canal no lineales. No obstante al no introducir los resultados del algoritmo base, estamos evitando información superflua (si aparece un correcto funcionamiento en los escenarios de igualación parece conveniente pensar que es correcto su funcionamiento en la implementación base), pero debemos añadir que estas simulaciones se han realizado, obteniéndose resultados satisfactorios, excepto en el algoritmo de Volterra, en el que nos introduciremos con mayor profundidad.

En este sentido, en el estudio previo de KRLS, realizaremos un recorrido por los tres montajes planteados en los experimentos, aunque sólo mostraremos resultados de la implementación base y del montaje de igualación no lineal sobre

canal estático, ya que son los resultados que ofrecen los autores del documento de referencia, y a partir de cuales vamos a comprobar el correcto funcionamiento de nuestra implementación de KRLS. El resto de resultados se comentarán en el apartado de comparativa entre todos los algoritmos. Con respecto a los algoritmos empleados para realizar la comparación de prestaciones, sólo comentaremos los parámetros que influyen en su configuración, ya que el entorno ya se habrá comentado con KRLS, y los resultados en la comparativa posterior.

## *5.2 Kernel Recursive Least Squares (KRLS)*

Para presentar el funcionamiento del algoritmo bajo estudio, podemos realizar un recorrido por las variables que disponemos en él, así como el significado que tienen. De está forma podremos tener un conocimiento más general del funcionamiento, lo cual nos permitirá ajustarlo mejor para obtener mejores resultados. La explicación de estos parámetros aparece en la Tabla 5.1 y Tabla 5.2.

| **Parámetro** | **Valor por defecto** | |
|---|---|---|
| $\nu$ | 0,01 | Se emplea al examinar la condición ALD[1], que no hace más que comprobar que el error de aproximación lineal se encuentra dentro de un determinado margen de validez. Ante una $\nu$ menor, introduciremos mas muestras en nuestro diccionario, con lo que incurriremos en menor error de aproximación, a costa de utilizar más memoria en el equipo, así como mayor tiempo de procesamiento, ya que cada en cada iteración el producto de cada nueva muestra por cada entrada al diccionario, debe llevarse al espacio destino. |
| $\lambda$ | 0,1 | Se emplea como margen cuadrático en el proceso de kernelización a través del cual llevamos las muestras del espacio de entrada, al espacio característico, es decir, en $k_{tt}$, y $\underline{\tilde{k}}_{t-1}\left(\underline{x}_t\right)$. A mayor valor tendremos mayor desajuste en el cálculo de la función kernel. |
| k | 2 | Nos informa del tipo de kernel que vamos a emplear en la operación de kernel. Los tipos de kernel, así como los parámetros que los gobiernan se pueden ver en la Tabla 5.2 |

**Tabla 5.1 – Parámetros de la implementación del algoritmo KRLS.**

[1] Del inglés “Approximate Linear Dependence”, que viene a significar dependencia lineal aproximada.

| Kernel | Valor 'k' | Valor por defecto | |
|---|---|---|---|
| Lineal | 0 | - | Definimos una función kernel que nos llevará los vectores del espacio de partida al espacio característico como el producto escalar. No dispone de parámetros. $$k\left(\underline{x},\underline{x}'\right)=\left\langle \underline{x},\underline{x}'\right\rangle$$ |
| Polinómico | 1 | a = 1<br>b = 1<br>c = 1 | En este caso la función del kernel se define con la siguiente expresión: $$k\left(\underline{x},\underline{x}'\right)=\left(b\left\langle \underline{x},\underline{x}'\right\rangle+c\right)^{a}$$ Tiene tres parámetros: a – grado del polinomio<br>b – constante multiplicativa<br>c – constante aditiva |
| Gaussiano | 2 | a = 1 | El kernel se construye mediante la expresión de una función de densidad de probabilidad gaussiana. $$k\left(\underline{x},\underline{x}'\right)=e^{-\frac{\left\|\underline{x},\underline{x}'\right\|^{2}}{2a^{2}}}$$ Tiene un parámetro: a – desviación típica del modelo |

**Tabla 5.2 – Tipos de kernel implementados para el algoritmo KRLS.**

Los valores por defecto que aparecen en la Tabla 5.1 y Tabla 5.2, se han obtenido realizando barridos manuales sobre simulaciones del algoritmo basadas en los resultados de los autores del documento de referencia de KRLS, y en el problema de igualación de canal no lineal. Las elecciones de estos parámetros, que finalmente serán los elegidos para las simulaciones finales, aparecen justificadas en los epígrafes siguientes: 5.2.1 y 5.2.2.

Los parámetros que principalmente deberemos considerar en nuestras simulaciones serán $\nu$, que controla muy directamente el comportamiento de nuestro test de dependencia lineal, el cual es la raíz del algoritmo. También deberemos tener especial cuidado con 'k' que nos permite elegir el kernel que utilizaremos en las transformaciones del espacio de entrada al espacio de salida. En función del escenario en el que nos encontremos y de la naturaleza del problema, deberemos elegir el kernel mas apropiado para nuestros intereses.

Una vez comentado como funciona nuestro algoritmo, y cuales son los parámetros que gobiernan su funcionamiento, solo nos resta hablar de las simulaciones que hemos realizado para corroborar la correcta ejecución del mismo. Para conseguir este objetivo vamos a basarnos en los resultados que ofrecen los autores del documento de referencia [Y. Engel, S. Mannor, R. Meir;2004]. Hemos de mencionar también que la implementación del algoritmo

que vamos a emplear en este caso es la más básica que hemos desarrollado, ya que el objetivo de este punto no es más que demostrar que las hipótesis que plantean los autores son ciertas y pueden llevarse a cabo de forma práctica.

Para poder decidir si nuestra implementación del algoritmo ratifica o no lo expuesto por los creadores del KRLS, disponemos de unas herramientas de comprobación. Para el funcionamiento del algoritmo disponemos de la implementación en código c del algoritmo base de los autores. De esta forma ejecutando su implementación base y la nuestra, sobre unos mismos datos de prueba conocidos, podemos determinar ambas implementaciones son equivalentes.

Con respecto a la aplicación del algoritmo en el problema de igualación de canal, disponemos de resultados numéricos presentados en el documento de referencia que usaremos como discriminadores entre ambas implementaciones.

### 5.2.1 Implementación base de KRLS

Como primer paso realizamos una primera implementación del núcleo de KRLS, tal y como se detalla en el documento de referencia [Y. Engel, S. Mannor, R. Meir;2004]. Para comprobar su correcto funcionamiento disponemos del código en lenguaje c de los creadores, el cual ejecutaremos con los mismos datos de entradas que para nuestra implementación.

Consideramos como salida del algoritmo el diccionario para el instante final, $\underline{\underline{D}}_t$. No obstante no nos será necesario devolver todos los elementos del diccionario, ya que son las entradas $\underline{x}_i$, que son conocidas al ser los mismos datos que estamos introduciendo. En este sentido nos bastará con devolver el vector de índices, $ii_t$ en el instante 't', y el número de elementos del diccionario $m_t$. Por otra parte, también consideraremos como resultado de la ejecución los coeficientes $\underline{\tilde{\alpha}}_t$, que nos permitirá recuperar la estimación de $\underline{\hat{y}}$.

Nos resta hablar de los datos que tomaremos como entrada para demostrar el funcionamiento similar de nuestra implementación con respecto a la versión en c de los autores. Dichos datos los tomaremos de un documento de los autores del algoritmo (detallaremos el documento, así como los datos de entrada en el apéndice A) en el que se detalla la forma de ejecutar la versión en c. Estos datos son reducidos, ya que solo se aplican 10 muestras, pero nos sirven para corroborar la validez de los resultados. Se realizaron mas simulaciones con otros datos mas extensos, pero no los mostramos por razones de espacio, y porque los resultados nos demuestran las mismas conclusiones. Ejecutando esos datos en ambas implementaciones obtenemos los resultados que aparecen en el resumen de la Tabla 5.3. Repetimos la simulación con tres conjuntos de parámetros para demostrar la importancia de la buena elección de estos, para obtener un conjunto de resultados eficiente.

| **Ejecución** | | **Implementación de los autores** | **Nuestra implementación** |
|---|---|---|---|
| Kernel gaussiano Parámetros por defecto (a = 1 $\nu = 0.01$) | $ii_t$ | 1 2 3 4 5 6 7 8 9 10 | 1 2 3 4 5 6 7 8 9 10 |
| | $\tilde{\alpha}_t$ | 24.255715 -4.369720 8.314685 -17.302834 -16.701088 2.845179 10.766788 -22.278816 -10.175265 4.421210 | 24.2558 -4.3697 8.3147 -17.3028 -16.7011 2.8452 10.7668 -22.2788 -10.1752 4.4211 |
| Kernel polinomico Parámetros a = 3 $\nu = 0.001$ Resto por defecto (b = 1 c = 1) | $ii_t$ | 1 2 3 4 5 6 7 8 9 10 | 1 2 3 4 5 6 7 8 9 10 |
| | $\tilde{\alpha}_t$ | 117.829565 42.756505 -45.768559 -208.037787 -17.520524 65.911536 3.698081 4.948576 -8.498704 -33.245941 | 117.8252 42.7691 -45.7661 -208.0244 -17.5194 65.8919 3.6979 4.9472 -8.4985 -33.2479 |
| Kernel polinómico Parámetros a = 3 b = 1 c = 0 Resto por defecto ($\nu = 0.01$) | $ii_t$ | 1 2 3 4 5 | 1 2 3 4 5 |
| | $\tilde{\alpha}_t$ | 407.377701 36.089453 -149.314193 -990.434224 -32.205324 | 407.3516 36.0896 -149.3009 -990.4187 -32.2050 |

**Tabla 5.3 – Tabla resumen de resultados de la implementación base de KRLS.**

Se han tomado estas tres posibles ejecuciones porque se detallan en el documento de los autores que mostramos en el apéndice A. No tienen nada de particular, no obstante nos sirven igual que cualquier otra para demostrar el funcionamiento equivalente. El enlace al código en c empleado por los autores se

encuentra en el apéndice A igualmente, así como las instrucciones para poder ejecutarlo.

Como podemos observar en la tabla, los resultados son similares, con lo que entendemos que ambas implementaciones son equivalentes.

Analizando lo obtenido, podemos determinar que el parámetro aditivo 'c', no tiene gran peso sobre el número de elementos del diccionario para la ejecución con kernel polinómico, ya que con c=0 en el cálculo de $\delta_t$ tenemos un valor más bajo que con c=1, lo cual determina nuestra elección $\delta_t >< \nu$ y por tanto la inserción o no al diccionario de la entrada en cuestión.

Otro factor a destacar es la diferencia existente entre el empleo del kernel polinómico y kernel gaussiano, ya que los pesos $\underline{\tilde{\alpha}}_t$ se incrementan bastante debido a la potenciación.

Por último, aunque no se percibe en los datos mostrados, pero si en el resto de simulaciones que hemos realizado, podemos notar la independencia con respecto al valor de $\nu$ de los $\underline{\tilde{\alpha}}_t$ (principalmente en el caso polinómico y sobre todo para grados altos). Esto se entiende, partiendo del hecho de que al potenciar con el parámetro 'a', los errores de dependencia lineal que detectamos en el test ALD de $\delta_t$ son elevados, por lo que obtendríamos los mismo valores que para un amplio rango de valores de $\nu$ (entre 0.001 y 1).

Con estas simulaciones consideramos que nuestra implementación es equivalente a la de los autores y estamos en disposición de aplicar el algoritmo en un problema más complejo.

Los autores del documento de referencia [Y. Engel, S. Mannor, R. Meir;2004], nos proponen varios experimentos sobre los que demuestran las cualidades del algoritmo. Entre ellos se encuentran regresiones no lineales, predicción de series temporales, igualación de canal no lineal y una comparación con procesos gaussianos reducidos. Nos vamos a centrar en el problema de igualación de canal no lineal, de manera reducida de momento, y a través de él queremos determinar las prestaciones del algoritmo, así como ratificar los

resultados que muestran los autores. Para ello comenzaremos a detallar un esquema de entrenamiento estático.

### 5.2.2 Esquema de igualación de canal basado en entrenamiento sobre canal estático

Nuestro primer paso es modificar nuestra implementación base del algoritmo ya comentada para ajustarla al problema de igualación de canal no lineal que se plantea en el documento de referencia. La modificación se centra en crear los datos que serán las entradas de nuestro KRLS. Por cada iteración se calculará la estimación y el error del algoritmo para calcular posteriormente la BER[2].

El montaje que construiremos en nuestra implementación del problema aparece en la Figura 5.1. La fuente de símbolos $u(t)$ no se define en el ejemplo del documento de referencia, por lo que tomamos una fuente aleatoria de símbolos BPSK {+1 -1}, de longitud M.

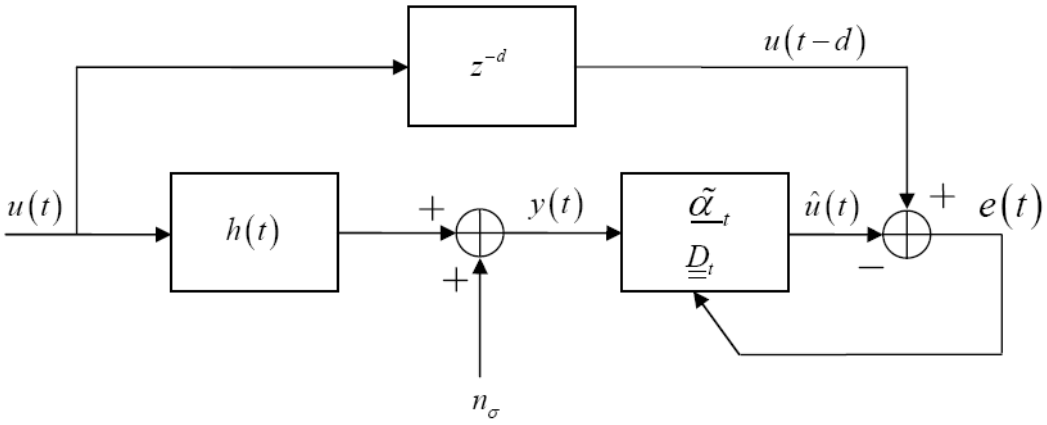



**Figura 5.1 – Esquema de igualación de canal implementado para KRLS**

Emplearemos el canal $h(t)$ que se detalla en el documento de los autores con orden 1 y cuya expresión aparece en la expresión 5.1, donde $n_\sigma$ representa el ruido AWGN añadido a la salida del canal. Este ruido se especifica en el documento con varianza 0.2.

$$\begin{aligned} x_t &= u_t + 0.5u_{t-1} \\ y_t &= x_t - 0.9x_t^3 + n_\sigma \end{aligned} \qquad \textbf{5.1}$$

[2] Trabajamos realmente con SER (Tasa de error de símbolo), pero es equivalente a la BER (Tasa de error de bit) en nuestra codificación BPSK.

El bloque $z^{-d}$ nos permite utilizar el retardo de igualación deseado a la hora de recuperar el símbolo de entrada $u(t)$. Este retardo será tratado posteriormente.

Los pasos que seguiremos en la preparación de los datos de nuestro problema serán los siguientes:

- Creación aleatoria de símbolos BPSK, $u_t$.
- Introducción de los símbolos creados por el canal lineal y obtención de $x_t$.
- Introducir la no linealidad a los datos de salida del canal lineal y sumar el ruido gaussiano $n_\sigma$, obteniendo de esta manera $y_t$
- Agrupamos los datos de salida en función de un nuevo parámetro, la longitud de suavizado, N, que se entiende como la anchura del filtro que usamos para recuperar nuestra entrada. En otras palabras, define como agrupamos los datos de entrada de nuestro KRLS de la manera que aparece en la expresión 5.2. Las filas de la matriz de dimensiones M-N x N serán las entradas de nuestro algoritmo. Para nuestro problema vamos a tomar N=2, debido a que el canal que vamos a emplear en nuestras simulaciones tiene dos elementos únicamente. En principio se podría usar cualquier otro valor de N y no afectaría a los resultados que mostraremos.

$$\underline{\underline{Y}}_t = \begin{pmatrix} y_1 & \cdots & y_{N+1} \\ \vdots & \ddots & \vdots \\ y_{M-N} & \cdots & y_M \end{pmatrix} \qquad \textbf{5.2}$$

- El último aspecto a tener en cuenta en nuestro problema de igualación de canal es el retardo que utilizaremos en nuestro esquema, ‘d’, que nos determinará que muestras $y_t$ tomaremos para recuperar nuestro $x_t$.

- Para nuestro problema en particular con longitud de canal 2, suavizado 2, entendemos que tendrían sentido las alternativas d=0,

d=1 y d=2, ya que contendrían algo de energía del símbolo de entrada $u_t$.

Comentado todo esto, estamos en disposición de comprobar el funcionamiento en el problema de igualación de canal planteado, comenzando con un entrenamiento de M=5000 símbolos independientes. Al ejecutar la simulación obtenemos los resultados mostrados en las figuras: Figura 5.2, Figura 5.3, Figura 5.4, Figura 5.5 y Figura 5.6 . La Figura 5.2 corresponde con el valor de los 5000 símbolos BPSK de entrada. La figura Figura 5.3 corresponde con la dispersión que introduce nuestro canal a los símbolos de entrada y la tercera gráfica corresponde con la dispersión que produce nuestro canal lineal y las distorsiones no lineales sobre nuestros agrupamientos con valor N=2. En la Figura 5.5 podemos ver la región de decisión que genera nuestro algoritmo KRLS tras el proceso de entrenamiento, y que determina la distinción de clases que se realiza. Por último, tenemos la Figura 5.6 en la que se observan las inserciones realizadas en el diccionario durante las 5000 muestras de entrenamiento.

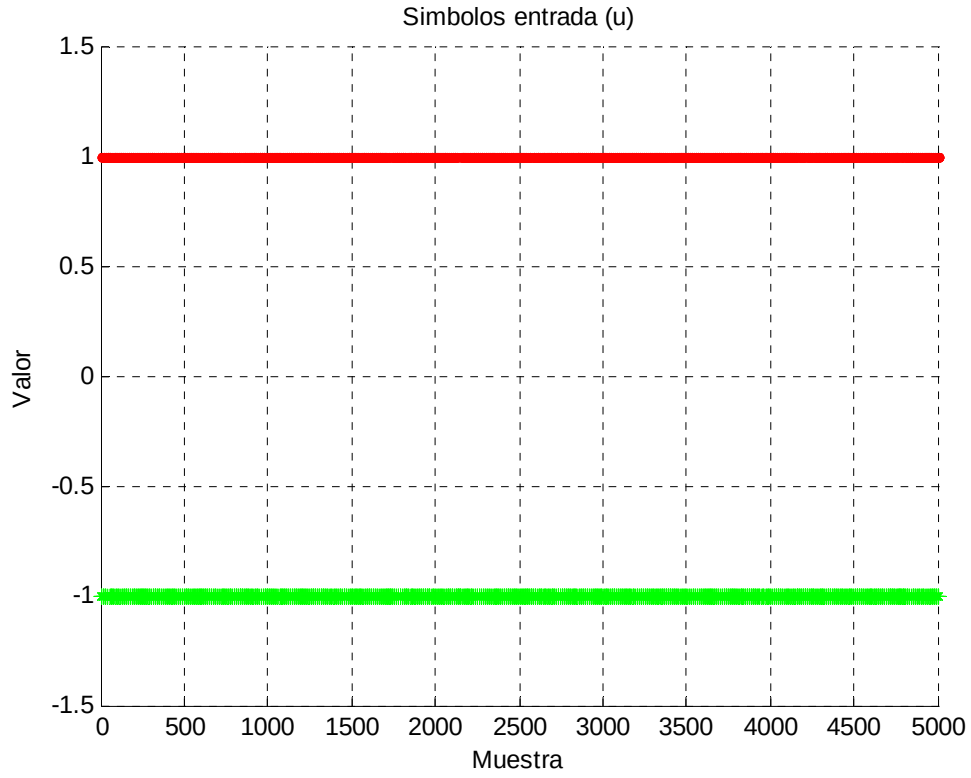



**Figura 5.2 – Símbolos generados independientes y equiprobables**

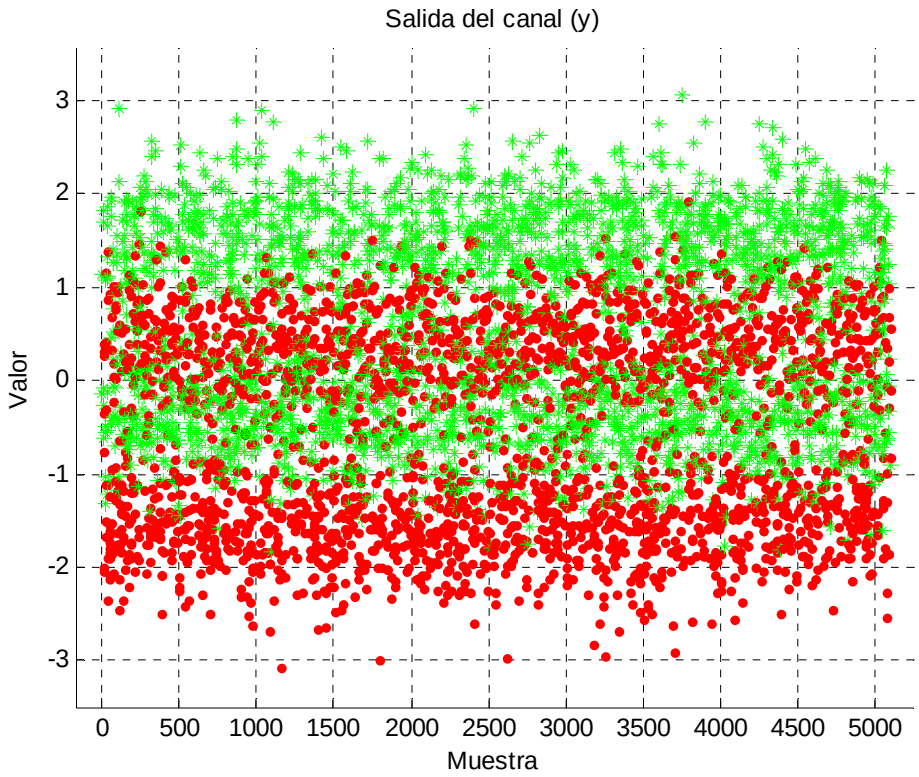



**Figura 5.3 – Símbolos de salida de nuestro canal**

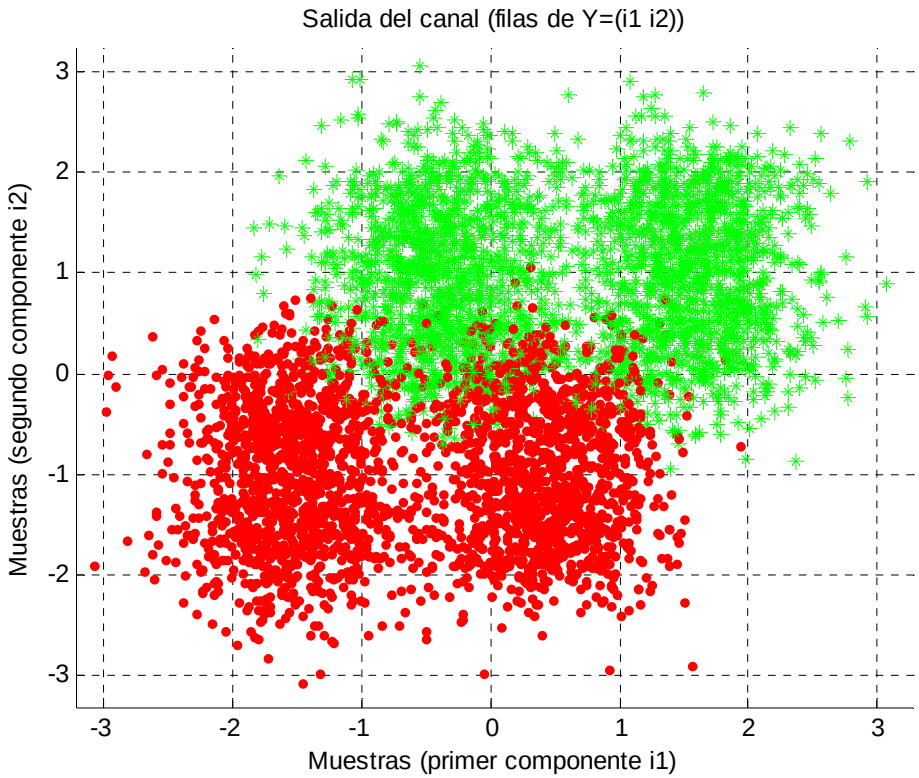



**Figura 5.4 – Símbolos de salida de nuestro canal agrupando los datos**

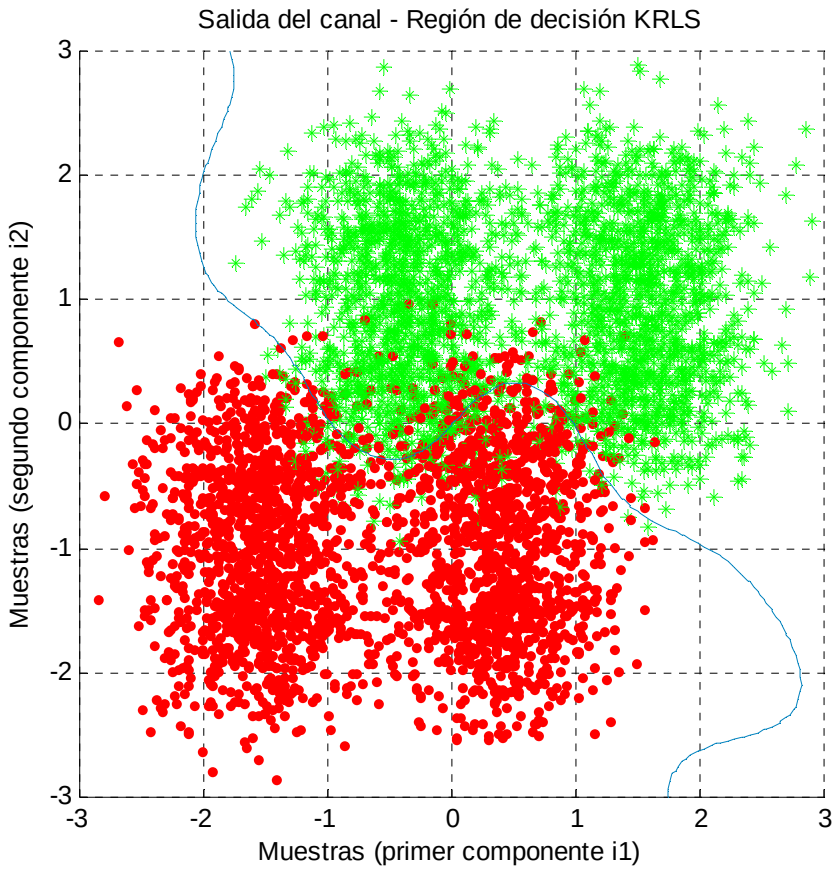



**Figura 5.5 – Región de decisión generada por el algoritmo KRLS**

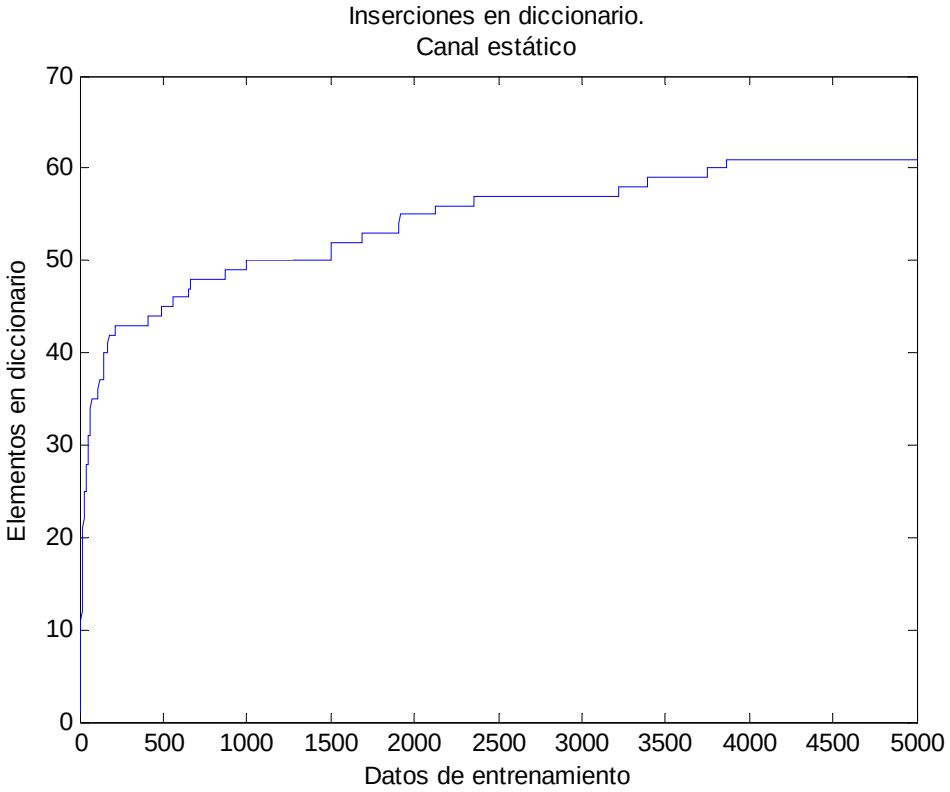



**Figura 5.6 – Inserciones en el diccionario del algoritmo KRLS**

Es común para todas las gráficas que los puntos rojos corresponden con los símbolos generados $u_t = +1$, y los asteriscos verdes a $u_t = -1$. Con respecto a la Figura 5.4, que tal y como hemos comentado corresponde con la ordenación de nuestros datos de salida en grupos de dos en relación a nuestra longitud de suavizado N=2 (en la gráfica mostramos las coordenadas que representan estas duplas de datos), debemos notar como separa y dispersa nuestro canal los símbolos de entrada para repartirlos en cuatro nubes de puntos. Estas nubes de puntos son constantes debido a que el canal que estamos usando es estacionario. Estos puntos son la entrada al algoritmo en ejecución y a partir de ellos recuperaremos nuestra señal generada $u_t$.

En la Figura 5.6 podemos ver que el algoritmo alcanza un estado estable en el tamaño del diccionario en 60 muestras aproximadamente para la elección de parámetros realizada en nuestra comparativa ALD.

Una vez comentada la generación de símbolos y su preparación, solo nos resta explicar como obtenemos la estimación del símbolo de entrada en el algoritmo KRLS, que es el que nos ocupa en el presente apartado, así como la mecánica de cálculo de la tasa de error de bit.

Tal y como comentamos en la explicación del algoritmo presente en el apartado 4.2 del presente documento, la forma de obtener la estimación del símbolo generado $u_t$ se podía realizar de dos formas alternativas: la primera de ellas correspondía con un procesado bloque de la información, mediante la expresión $\hat{\underline{y}} = \underline{\underline{A}}_t \underline{\underline{\tilde{K}}}_t \underline{\tilde{\alpha}}_t$. La segunda alternativa enfoca el problema desde un punto de vista online y en tiempo real con la expresión del algoritmo que aparece en 5.3.

$$\hat{\underline{u}}_t = \underline{\tilde{k}}_t \left( \underline{Y}_{t-d} \right)^T \underline{\tilde{\alpha}}_t \qquad \textbf{5.3}$$

Hemos de notar la diferencia de notación entre lo expuesto en el apartado de salida correspondiente al epígrafe 4.2.2.4, y lo que aparece en la expresión 5.3 con respecto a la estimación online. La diferencia de notación radica en que en el ejemplo práctico en el que nos encontramos, tenemos que la entrada pasa a ser las filas de la matriz $\underline{\underline{Y}}$, en lugar de $\underline{x}$, y la estimación es $\hat{u}$ frente a la anterior $\hat{y}$.

Además, con el objetivo de modelar nuestro problema, hemos introducido en la expresión un retardo de igualación.

Para conseguir uno de los objetivos de nuestro problema, hemos optado en la implementación por elegir la estimación en tiempo real, trabajando de forma online.

Con respecto al cálculo del error del algoritmo, lo calculamos iteración a iteración, y viene a significar el error que cometeríamos de usar los pesos calculados hasta el momento con una nueva entrada no contemplada aún. En términos más matemáticos, podríamos decir que intentaríamos realizar la estimación $\hat{u}_t$ con los pesos referentes a $\hat{u}_{t-1}$. La expresión del error que hemos empleado la podemos ver en 5.4.

$$e_t = u_t - \hat{u}_t^{t-1} = u_t - \underline{\tilde{k}}_{t-1}\left(\underline{Y}_{t-d}\right)^T \underline{\tilde{\alpha}}_{t-1} \qquad \textbf{5.4}$$

En lo referente al cálculo de la BER, debemos decir que al emplear unos datos correspondientes con una constelación BPSK, tendremos que la BER será igual a la SER. Para poder calcularla, introducimos un sencillo detector de signo sobre la muestra estimada, $\hat{u}_t$, obteniendo la muestra detectada, $\hat{u}_t^d$. Esto que comentamos aparece en la expresión 5.5.

$$\hat{u}_t^d = sign\left(\hat{u}_t\right) \qquad \textbf{5.5}$$

Entendemos como reducción porcentual en nuestro algoritmo como la relación en tanto por ciento que existe entre el número de elementos del diccionario, $m_t$, y el número total de muestras de entrenamiento, 'M'. Apoyados en esta definición calcularemos la relación porcentual de las ejecuciones que realicemos de nuestro algoritmo.

Para comprobar el funcionamiento de nuestra implementación de KRLS en el problema estático de igualación no lineal de canal, vamos a basarnos en los resultados experimentales que muestran sus autores en el documento de referencia [Y. Engel, S. Mannor, R. Meir;2004], ya que la situación que plantean es similar a la nuestra.

Tomando esta información como base, debemos tener en cuenta que los autores del algoritmo realizan simulaciones de entrenamiento con 5000 símbolos aleatorios, y realizando un estudio estadístico (obtienen el cálculo de la media y de la desviación típica) con 100 repeticiones independientes. El kernel que emplean corresponderá con el caso polinómico de grado ocho. Con respecto a la elección de parámetros, los autores, eligieron $\nu = 0.1$.

En nuestro caso, vamos a utilizar 5000 símbolos BPSK aleatorios, pero no vamos a realizar el análisis estadístico de los resultados, ya que esta comprobación no deja de ser un paso previo a los verdaderos resultados que mostraremos más adelante. Con respecto al kernel empleado y a la elección del parámetro $\nu$, comentaremos más adelante nuestra configuración.

Una vez comentados todos estos aspectos, podemos ver los resultados de las simulaciones en la Tabla 5.4, en la que hemos realizado una ejecución por cada retardo de igualación posible con un índice de suavizado de N=2, como es nuestro caso. Además en la tabla, podemos comparar nuestros resultados, con los aportados por los autores del algoritmo.

| Retardo de igualación | | Resultados de referencia | Nuestros resultados |
|---|---|---|---|
| d = 0 | BER | 0.173±0.010 | 0.2923 |
| | Reducción (%) | 9.7±0.6 | 9.9 |
| d = 1 | BER | 0.065±0.006 | 0.1102 |
| | Reducción (%) | 9.7±0.6 | 10.06 |
| d = 2 | BER | 0.046173±0.004 | 0.0412 |
| | Reducción (%) | 9.6±0.7 | 10.06 |

**Tabla 5.4 – Resultados KRLS sobre esquema de igualación con entrenamiento estático.**

Como podemos ver en los resultados mostrados, obtenemos unos resultados equivalentes en el mejor retardo de igualación que tenemos en nuestro esquema, d=2. Para los otros dos retardos, los resultados son mas dispares, siendo peores en prestaciones de BER, aunque no en reducción del tamaño del diccionario. Hemos de determinar que siendo el retardo que elegiremos en las

próximas simulaciones d=2, entendemos que los resultados aquí mostrados serán válidos[3] y nos permitirán avanzar en nuestras conclusiones.

Con respecto a los datos mostrados en la Tabla 5.4, no hemos empleado la misma ejecución que realizan los autores en su documento (kernel polinómico de grado 3 con $\nu$=0.1 con coeficiente multiplicativo 1 y coeficiente sumatorio 0), ya que los resultados que obteníamos eran muy pobres con respecto a los suyos. Por esta razón, y apoyándonos en la afirmación que realizan los autores de que no han realizado otra simulación aparte de la comentada, realizamos otras simulaciones con el objetivo de obtener unos resultados lo mas parecidos posible a los suyos. En este sentido hemos mostrado los datos en la tabla con una ejecución basada en un kernel gaussiano de varianza 0.2 y $\nu$=0.1. Otra diferencia a simple vista, es que los autores realizan un promedio estadístico sobre 100 realizaciones independientes. Es evidente que nosotros no hemos realizado esta labor, de manera intencionada, ya que este paso lo entendemos como una acción previa para conseguir los resultados siguientes de entrenamiento, tal y como hemos comentado anteriormente.

Una posible explicación sobre la discrepancia entre los resultados, junto con la que acabamos de comentar relacionada con el kernel empleado, puede ser la diferencia de la fuente de datos empleada, ya que desconocemos como generan la información, siendo nuestro único punto de apoyo los resultados mostrados y el canal empleado.

Con respecto a la Tabla 5.4 podemos distinguir también que el número de valores en el diccionario se mantiene constante con independencia del retardo de igualación usado. Esto es debido a que al ser el canal estacionario, las nubes de puntos se mantienen en una posición constante, lo que produce que con un número bajo de entradas en el diccionario tomadas al iniciar el algoritmo baste para representar a todos los puntos (el mínimo de entradas del diccionario sería $m_t$=4, al existir cuatro grupos de puntos). Como consecuencia de esto tenemos que la reducción en tanto por ciento va descendiendo en función de que la longitud de datos se incremente (‘M’ mayor con ‘m’ constante).

[3] En la situación de trabajar con retardos d=1 o d=0, tendríamos una discrepancia mas significativa con la implementación de los autores.

Una conclusión a la que hemos llegado en nuestras simulaciones (aparte poder que tiene la longitud del diccionario en la tasa de error que obtenemos[4]), es que modificando nuestro kernel gaussiano, reduciendo la varianza del modelo de 0.2 a 0.1, hemos realizado de nuevo las simulaciones, obteniendo un valor de SER[5] = 0.0240 con una tasa de reducción del 30.8%. En estos resultados, podemos comprobar que al reducir nuestra varianza hemos llegado a aumentar el tamaño de nuestro diccionario, lo cual implica que tenemos un mejor ajuste de nuestros datos dando lugar a una tasa de error mucho menor. En definitiva no podemos considerar este resultado como más satisfactorio que el mostrado en la tabla anterior, ya que aunque la tasa de error de símbolos es significativamente menor, nos encontramos con que el gasto de memoria correspondiente al diccionario se ha incrementado 3 veces, aumentando consecuentemente el tiempo de proceso del algoritmo.

En general los resultados de BER que mostramos para el caso del canal no lineal y estacionario, son bastante malos, incluso para el caso de d=2. No obstante eso es normal, ya que desde el punto de vista de igualación de canal, nuestro objetivo es minimizar la BER, mientras que nuestro punto de partida para el KRLS ha sido el MSE. Teniendo en cuenta esto, podemos entrar a valorar el uso de este algoritmo de aprendizaje para igualación de canal.

Con respecto a la comparativa entre nuestros resultados y los de referencia, tenemos que hacer varios apuntes. Primero, en el documento de referencia de los autores se expone como resultado la velocidad en segundos de la ejecución del algoritmo. Nosotros no hemos tenido en cuenta ese factor, ya que es muy dependiente del lenguaje de implementación utilizado[6].

Por último para comprender mejor el funcionamiento del algoritmo, realizamos una simulación con cada retardo expuesto, para poder ilustrar mejor el comportamiento de la simulación. En los resultados mostramos la señal recuperada frente a la enviada (para visualizar esto, hemos centrado nuestra atención en el instante final de la simulación, correspondiente al intervalo de

[4] Debemos tener en cuenta que un mayor tamaño del diccionario implica un mayor tiempo de proceso, ya que deberemos comparar cada nueva entrada al sistema con cada uno de los elementos del diccionario.
[5] En nuestra simulación tenemos que BER es semejante a SER, ya que tenemos una fuente de datos BPSK.
[6] En nuestro caso hemos realizado la simulación en MatLab, mientras que los creadores la realizaron en C.

muestras entre la 4900 y la 5000), así como el error del algoritmo frente al instante en que se introduce otra nueva entrada al diccionario.

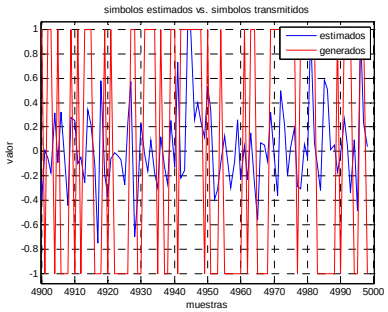
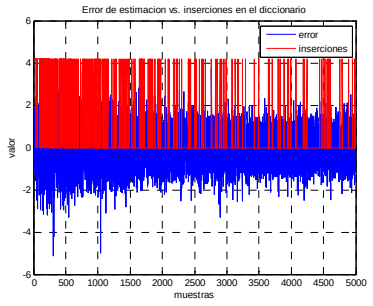
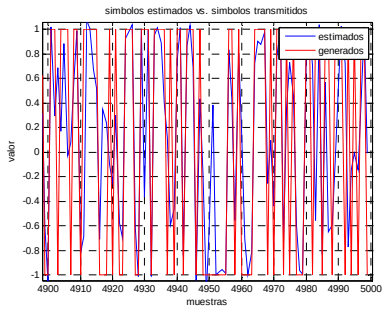
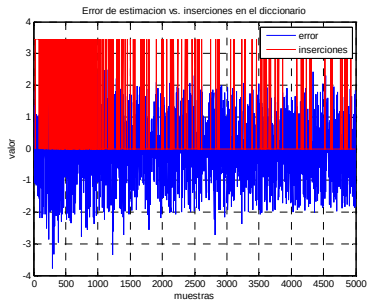
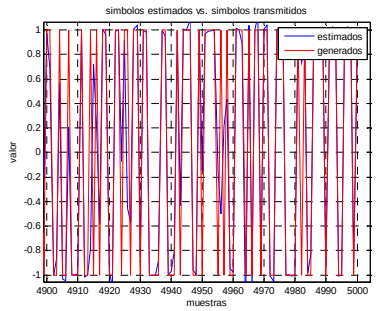
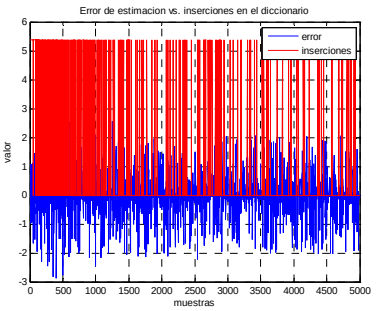
| Retardo de | Señal recuperada Vs. Señal generada | Error Vs. Inserciones en el diccionario |
|---|---|---|
| d = 0 |   |   |
| d = 1 |   |   |
| d = 2 |   |   |

**Figura 5.7 – Gráficas de estimación y de inserciones en el diccionario**

En las gráficas que mostramos en la Figura 5.7 podemos ver un menor error en el retardo d=2 que se va amplificando según reducimos el retardo de igualación. Esto se ve apoyado con las gráficas de símbolos estimados frente a los símbolos transmitidos, ya que se evidencia una peor estimación y seguimiento de los símbolos estimados según reducimos el retardo. Esto refuerza los resultados mostrados en la Tabla 5.4. Con respecto a la reducción del error que hemos comentado y que aparece en las gráficas de error de estimación frente a inserciones en el diccionario, podemos ver que dicha reducción no se aplica sobre

el máximo del error, que se mantiene en los tres casos con una cota superior de ±2, sino en el número de muestras que presentar error, que puede determinarse con la densidad de error que aparece en dichas gráficas. También podemos ver que la cota máxima de error se mantiene constante en todos los casos, ya que no depende del retardo de igualación elegido, sino de la fuente de los datos y de los parámetros de ejecución de nuestro algoritmo. También podemos observar como esta cota máxima de error se alcanza después de haber introducido muestras en nuestro diccionario. Una vez se han realizado estas inclusiones, tenemos un escenario estacionario que nos permite alcanzar esta cota fija de error. Como tal, este error que mostramos está muy relacionado con el número de elementos en nuestro diccionario, ya que si no tenemos suficientes elementos en él, no seremos capaces de establecer una relación lineal lo suficientemente buena como para crear las siguientes entradas a nuestro sistema. Por esta razón nuestro error se dispara al comienzo del algoritmo, y se reduce cuando tenemos un diccionario mas poblado.

En nuestras simulaciones hemos podido ver también, como nuestra estimación se ajusta perfectamente a los símbolos transmitidos mientras estamos introduciendo datos en nuestro diccionario, y que cuando dejamos de introducirlos y empezamos a aproximar es cuando vemos diferencias evidentes entre ambas señales.

Con respecto a la gráfica de error podemos explicar los picos que aparecen como consecuencia de propio funcionamiento de nuestro algoritmo, ya que corresponden con un $\underline{\tilde{\alpha}}_{t-1}$ elevado o con un $\underline{\tilde{k}}_{t-1}\left(\underline{Y}_{t-d}\right)$, tal y como podemos discernir de la expresión del error dada en este mismo apartado. Por visualización de los resultados hemos podido determinar que lo que provoca los picos principalmente es un elevado valor en el vector $\underline{\tilde{k}}_{t-1}\left(\underline{Y}_{t-d}\right)$.

Para finalizar esta parte del documento, vamos a realizar una comparación entre los kernels que hemos implementado, eligiendo un retardo d=2 y realizando simulaciones con 5000 muestras y $\nu$ =0.1.

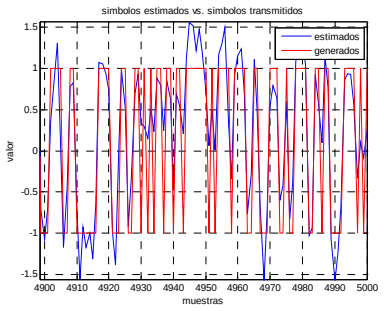
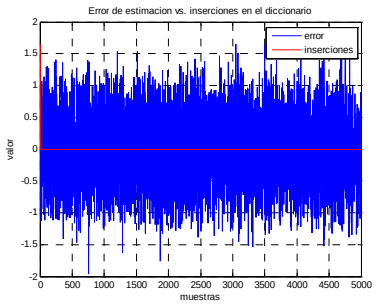
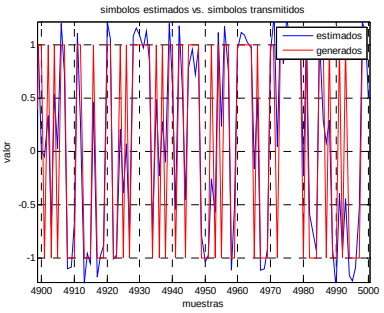
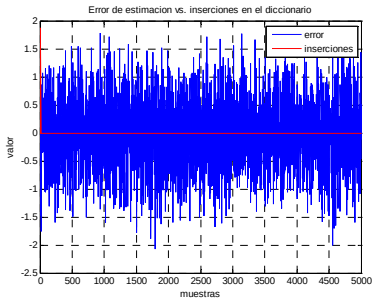
| Kernel utilizado | Señal recuperada Vs. Señal generada | Error Vs. Inserciones en el diccionario |
|---|---|---|
| Lineal |  |  |
| Polinómico |  |  |

**Figura 5.8 - Gráficas de estimación y de inserciones en el diccionario para kernel lineal y polinómico**

Para las gráficas mostradas en la Figura 5.8 hemos realizado las siguientes ejecuciones: para la alternativa lineal no tenemos ningún parámetro en nuestro kernel que modele su funcionamiento; para la segunda alternativa, el caso de kernel polinómico, hemos optado por elegir un kernel de grado 3 con coeficiente multiplicativo 1 y coeficiente sumatorio 0, siendo equivalente esta ejecución a la que muestran los autores en el documento de referencia.

Los resultados de error que hemos obtenido en estas simulaciones son los siguientes: para el caso lineal tenemos una BER de 0.0808 con una tasa de reducción de 0.004%; para el caso polinómico de orden 3, tenemos una BER de 0.0668 y una tasa de reducción de 0.2%. Hemos de evidenciar que tener unas tasas de reducción tan bajas[7] han supuesto que los tiempos de ejecución del algoritmo sean muy reducidos en comparación con los resultados mostrados en la Tabla 5.4.

[7] Para el caso lineal solo se introducen en el diccionario 2 inserciones frente a las 5000 muestras que se introducen al sistema. Para el caso polinómico de orden 3, se introducen 10 muestras en el diccionario sobre el mismo número de muestras totales.

Como podemos ver en los resultados numéricos que acabamos de detallar, el funcionamiento de estos dos kernels para una d=2, si bien no llegan a los resultados mostrados usando el kernel gaussiano, si producen una salida satisfactoria en comparación, ya que la tasa de reducción es muy reducida, al igual que los tiempos de ejecución.

En las gráficas que aparecen en la Figura 5.8 podemos ver que la aproximación con el kernel lineal de nuestra señal estimada frente a la señal generada no es demasiado limpia. Da buenos resultados, pero el rango de variación del algoritmo es lento, provocando errores en señales con variación alta como es el caso de nuestra fuente binaria (el error al que hacemos referencia es el error de seguimiento a una señal binaria tal y como aparece en la gráfica de símbolos estimados frente a símbolos transmitidos). Para permitir variaciones más rápidas, debemos recurrir al kernel polinómico, siendo más variable cuanto más alto sea el grado (aunque esto provocará que los picos de error sean más elevados igualmente).

En principio, tal y como hemos mencionado, en nuestra ejecución polinómica, que es semejante a la que realizan los autores, no hemos conseguido llegar a los resultados que ellos muestran (ellos ofrecían una BER de 0.046 ± 0.004, y nosotros hemos obtenido 0.0668). En este sentido, no hemos podido replicar completamente su entorno de trabajo, por esa razón elegimos un kernel gaussiano para nuestros resultados mostrados en la Tabla 5.4, aunque eso suponga un menor grado de reducción. Esta diferencia de resultados la achacamos principalmente al desconocimiento de la fuente de datos empleada por los autores.

Con respecto al parámetro del kernel gaussiano, se realizó un barrido manual, mediante el cual establecimos que el valor óptimo para nuestro caso es el elegido como por defecto. En este punto se garantizó un valor cercano a la varianza de la fuente de datos empleada (cuyo valor es 2) y un valor realista para nuestros propósitos. Por esa razón se justifica la elección de a = 1.

Por todo lo dicho consideramos que nuestros resultados son correctos, ya que suponen un entorno de trabajo diferente. Esto nos abre la puerta a avanzar hacia el siguiente paso de nuestro trabajo, realizar implementaciones y

simulaciones sobre un canal dinámico para comprobar el comportamiento del algoritmo ante variaciones del canal no lineal.

### 5.2.3 Esquema de igualación de canal basado en entrenamiento sobre canal dinámico

Una vez comentados nuestros experimentos sobre un esquema de igualación con entrenamiento estático, vamos a proceder a extenderlo hacia un entorno dinámico, en el que nuestro canal deje de comportarse de manera constante.

El objetivo de este punto, es el de realizar un estudio de la capacidad de seguimiento de nuestro algoritmo no lineal KRLS ante variaciones del canal de comunicaciones. Tal y como comentamos en la introducción del presente proyecto fin de carrera, sabemos que la capacidad de enganche de los algoritmos adaptativos es muy importante, ya que produce que nuestras estimaciones sean fiables en entornos variantes, o con desvanecimiento, como pueden ser por ejemplo, los sistemas de telefonía móvil.

Para realizar nuestros experimentos, vamos a modificar la implementación del sistema que mostramos en el apartado anterior, y que aparece en la Figura 5.9, en la que nos encontramos ante un escenario estático con un canal constante. Para abordar ese problema realizamos un entrenamiento continuo en el que obteniamos la toma de datos de error y reducción al finalizar la entrada de datos de entrenamiento al sistema.

La alternativa que se plantea en este apartado es la que aparece en la Figura 5.10. En este sistema, tenemos un entorno dinámico en el que aparece en escena una variación de canal de cualquier tipo. No nos sirve el montaje estático para analizar esta nueva situación. Por esta razón se plantea la inclusión de muestras de test generadas desde la misma fuente de datos de entrenamiento que evalúan nuestro algoritmo durante el proceso de entrenamiento.

En este esquema, nos encontramos con que nuestro entrenamiento debe adaptarse a la variación de canal, ya que este es variante. En este sentido, debemos generar nuestros datos de test para el instante ‘k’ con el canal correspondiente a ese mismo instante, es decir, $\underline{h}_k$ . Estos datos de test nos darán la información de prestaciones del entrenamiento de nuestro algoritmo. De esta

forma tendremos una manera de conocer como se comporta nuestro algoritmo bajo entornos dinámicos, en aspectos tales como: velocidad de enganche a la nueva situación, máxima variación permitida en el canal, etc.

La implementación que realizamos comienza con un entrenamiento base de nuestro algoritmo. Permitimos inicialmente que nuestro canal sea constante para que nos encontremos en un esquema de entrenamiento estático. De esta manera nos aseguramos que trabajaremos sobre un algoritmo que se encuentra enganchado a la fuente de datos de entrada. Establecemos un estado de nuestra ejecución que converge a nuestra situación estática. Una vez alcanzado este primer objetivo, modificamos la ejecución de nuestro algoritmo para que trabaje muestra a muestra. En este punto debemos modificar nuestra fuente de datos para que genere datos de entrada a nuestro sistema con una variación de canal establecida[8]. Estos datos serán los que alimentarán nuestro sistema dinámico. Cada cierto tiempo, que elegiremos como parámetro de nuestra ejecución, nuestra fuente de datos generará un bloque de datos de test, de longitud también parametrizable, que nos permitirá comprobar como se encuentra el estado de nuestro algoritmo (el estado lo denotamos con $\underline{w}_k$, que entendemos como los pesos de nuestro algoritmo en particular) en función del número de muestras de entrenamiento utilizadas, para el instante 'k'. De esta manera podremos seguir la evolución de nuestra ejecución y de nuestro entrenamiento.

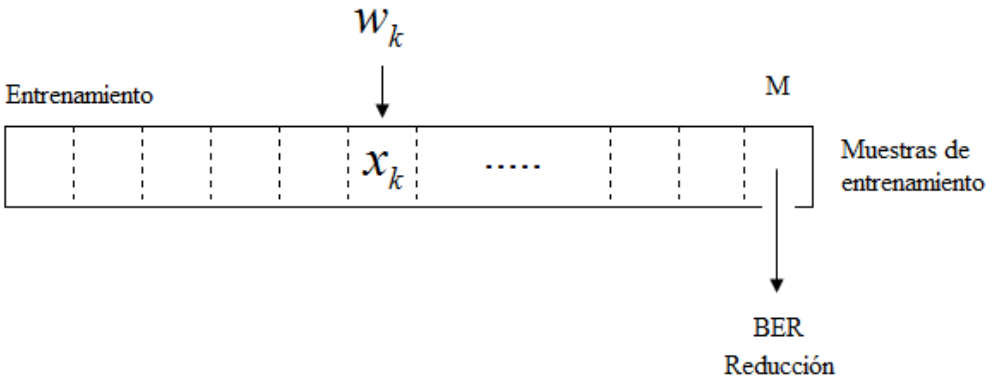



**Figura 5.9 - Esquema de implementación del entorno estático**

[8] Nuestra fuente de datos también debe ser capaz de generar bloques de datos de test que correspondan con el canal correspondiente al instante de tiempo en el que se encuentre la ejecución.

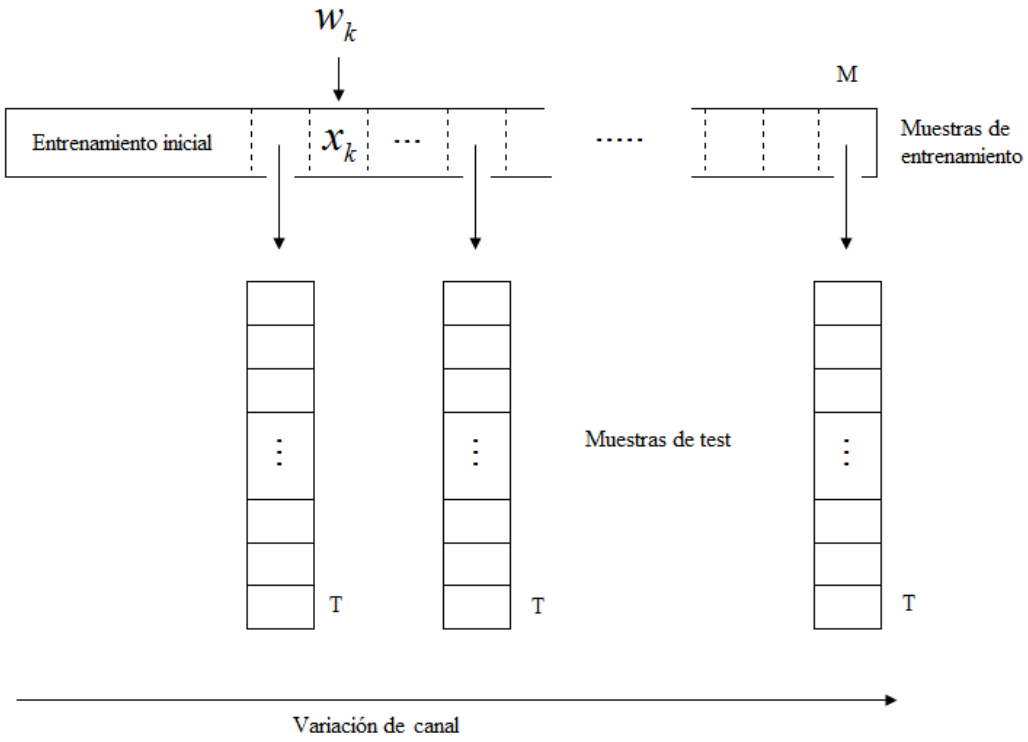



**Figura 5.10 – Esquema de implementación del entorno dinámico**

Un factor importante que debemos tener en cuenta es la longitud,'T', de las secuencias de test, ya que van a determinar la veracidad de los datos que obtengamos en nuestras gráficas. En general, las secuencias de test deben tener al menos una década más que la inversa de la mínima tasa de error que alcance nuestro algoritmo. Si se garantiza este aspecto, tendremos que los resultados que mostramos serán correctos.

Otro factor a tener en cuenta es la longitud de entrenamiento, ya que tiene una relación directamente proporcional con el tiempo que dispone nuestro algoritmo para adaptarse a la nueva situación. Otro aspecto importante es la variación del canal. Un cambio demasiado abrupto o elevado, puede provocar que nuestro algoritmo no pueda converger a la nueva solución.

Por último, tenemos que tener en cuenta otro aspecto, menos importante en comparación con los anteriores. Es el espaciado de los grupos de test dentro de la secuencia de entrenamiento. Esto vendrá determinado por la variación del canal, por la precisión que se desee en los resultados, etc. Un espaciado demasiado pequeño puede provocar que los tiempos de proceso se disparen.

En este apartado vamos a probar cuatro tipos de canales que nos permitirán determinar el nivel de adaptación que tiene nuestro algoritmo: un canal con una variación senoidal muy baja, senoidal con mayor variación, señal escalón de baja altura y por último un escalón con mayor amplitud[9]. De esta manera, con estos canales, perseguimos estudiar el comportamiento del algoritmo y su nivel de recuperación ante cambios en el comportamiento del canal. Debemos añadir que el canal inicial que usaremos será el mismo que en el caso estático, y que no se modificará hasta que el algoritmo haya convergido a la solución correcta. A partir de este punto será cuando introduzcamos nuestras variaciones. Hemos elegido estas variaciones por considerarlas señales básicas que pueden ser encontradas en forma de señal de error en cualquier sistema de comunicaciones, si bien los valores de amplitud de estas señales se han elegido de manera completamente arbitraria con el único fin de comprobar las prestaciones del algoritmo.

Inicialmente vamos a comenzar con una simulación con un canal estático en todo el espacio de entrenamiento, pero basándonos en el montaje de la Figura 5.10 en lugar del propuesto para estático en la Figura 5.9. La razón de este cambio de escenario se encuentra motivada porque con el nuevo montaje, vamos a conseguir gráficas más interesantes, ya que introducimos las estimaciones con las muestras de test. En este sentido vamos a mostrar dos gráficas de funcionamiento del algoritmo: SER frente a SNR, en la que realizaremos una comparación de nuestro algoritmo KRLS frente a los algoritmos contrincantes. La segunda gráfica que mostraremos será de patrones frente a muestras de entrenamiento. En ella, nuestra intención será la de mostrar la evolución del ciclo de entrenamiento del algoritmo a través de las estimaciones que se realizan con las muestras de test. De esta forma, tendremos que por cada cierto número de muestras de entrenamiento, evaluaremos nuestro conjunto de test que nos permitirá establecer un nuevo valor en la gráfica de patrones frente a muestras. Al igual que en el caso de la gráfica de SER frente a SNR, vamos a comparar nuestra gráfica de patrones de KRLS con el resto de algoritmos en el epigrafe de comparativa del presente proyecto fin de carrera.

[9] En la comparativa de los resultados comentaremos más en profundidad las señales de variación que hemos empleado.

### 5.2.4 Esquema de igualación de canal basado en entrenamiento guiado por decisión

En este apartado, vamos a modificar el escenario anterior para realizar un experimento en entrenamiento guiado por decisión. Esto supone una modificación en el concepto del anterior punto, ya que ahora introducimos un nuevo matiz en la mecánica de entrenamiento del sistema. Volvemos a tener inicialmente una pequeña etapa de símbolos de entrenamiento, al igual que en el caso dinámico anterior. Una vez finalizado este entrenamiento inicial, tenemos que los propios símbolos que empleamos para entrenar nuestro sistema, los estimamos. Por explicarlo de otro modo, podemos decir que usamos para entrenar el sistema los mismos símbolos que estimamos anteriormente con nuestro sistema.

De forma gráfica podemos ver el esquema en la Figura 5.11 el montaje de la nueva implementación. En ella aparece que para continuar el entrenamiento en el instante N+1 usamos la dupla $(\hat{x}_{k+1}, y_{k+1})$. En este conjunto de datos, tenemos que $\hat{x}_{k+1}$ se obtiene de $y_{k+1}$ que es conocida, y del entrenamiento anterior, del que la última muestra que se conoce es la correspondiente al peso $w_k$ con la dupla $(x_k, y_k)$, que pertenece al entrenamiento inicial del escenario. De esta manera se continúa el entrenamiento del sistema basándonos en las propias estimaciones de símbolos de entrada que realizamos en cada iteración. El símbolo estimado de salida en el instante k+1 corresponderá a los pesos $w_{k+1}$, no a $w_k$ que son utilizados para la estimación previa de $\hat{x}_{k+1}$ en el entrenamiento.

Un factor importante a tener en cuenta en este escenario, es que ante situaciones con gran cantidad de errores, estos se pueden propagar fácilmente, ya que al realizar mal las estimaciones de los símbolos que se emplean para entrenar el algoritmo, realizamos un entrenamiento erróneo que se propaga rápidamente en los pesos que utilizamos para estimar los símbolos de entrada en las iteraciones siguientes.

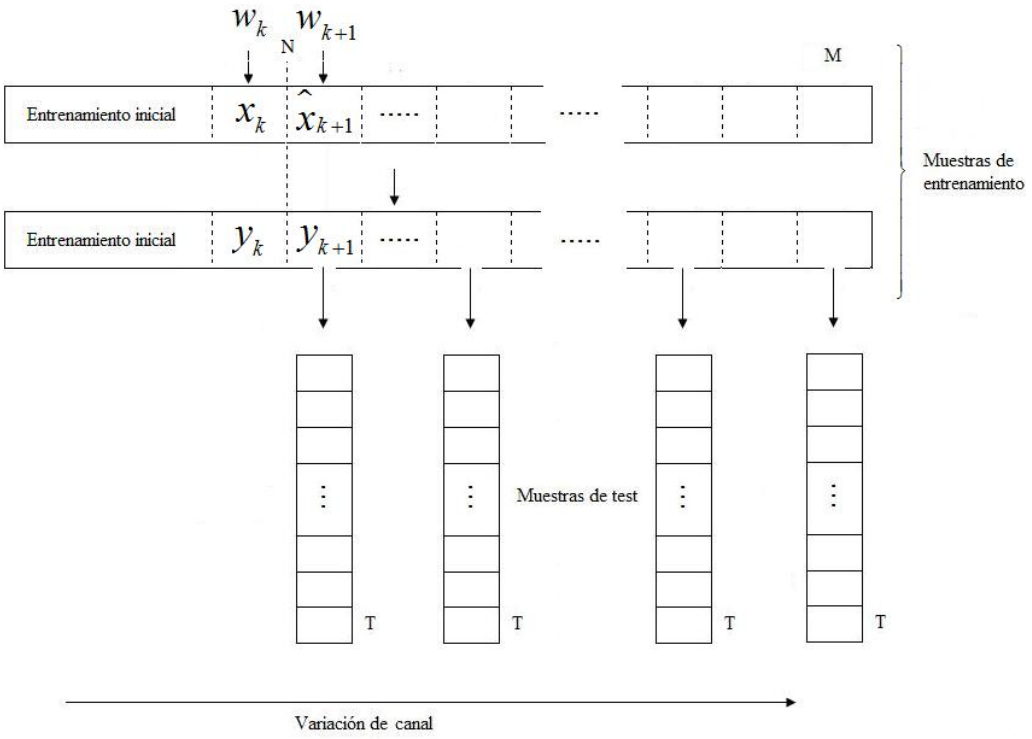



**Figura 5.11 – Esquema de implementación del entorno dinámico entrenado por decisión**

Una variación de canal demasiado rápida o muy abrupta puede provocar que nuestro algoritmo no encuentre la solución adecuada y oscile, ya que que todas las estimaciones anteriores carecerían de sentido, lo cual implica que todo el entrenamiento anterior nos haga incurrir en error en lugar de ayudarnos a evitarlo. Está muy relacionado con el concepto de memoria de nuestro sistema. Esta problemática aparecía en el esquema de entrenamiento anterior, pero en este caso toma más fuerza, al ser un entorno sin supervisión.

Para probar el comportamiento del escenario vamos a obtener gráficas utilizando las mismas cuatro variaciones de canal empleadas en el escenario dinámico, con la salvedad de que en este caso, además de medir el nivel de adaptación, nos proponemos a medir la propagación de errores del entrenamiento basado en decisión de forma adicional.

En este sentido nos proponemos introducir además de las gráficas de BER frente a SNR y de patrones durante el entrenamiento, la gráfica correspondiente al error cuadrático, que nos va a justificar la convergencia del algoritmo en su estado inicial y la posible pérdida de ella en los cambios bruscos de señal.

## *5.3 Clasificadores KNN*

Al igual que hicimos en el caso del algoritmo KRLS, vamos a realizar un recorrido sobre los parámetros que controlamos en la simulación de nuestra implementación del algoritmo basado en el documento [P.Savazzi, L. Favalli, E. Costamagna, A. Mecocci; 1998] que emplea los clasificadores KNN en un problema de igualación de canal no lineal práctico en un entorno móvil GSM. En nuestro caso, tal y como hemos comentado anteriormente, nos centramos únicamente en la aplicación que realizan los autores del algoritmo, y no en el problema en sí que plantean en el documento. Hemos tomado el algoritmo y lo hemos adaptado a nuestro marco de trabajo para el presente proyecto.

Una vez comentado esto, podemos ver los parámetros que rigen el comportamiento:

- 'K': Es el parámetro principal en este algoritmo, ya que va a determinar el número de vecinos necesarios en el proceso de agrupamiento de clases, necesarios para catalogar una nueva muestra de entrada. Si le damos un valor alto, aumentaremos el tiempo de procesamiento, pero en cambio tendremos que la decisión sobre la nueva muestra habrá sido construida bajo una base de información mayor, por lo que encontraremos una menor probabilidad de error.
- Mapa knn: Realmente no es un parámetro del sistema, aunque en implementaciones posteriores lo trataremos como tal, al considerarlo como la inicialización del algoritmo. No es más, que una librería de entradas catalogadas que nos servirán para determinar las clases de los 'k' vecinos más próximos. En este caso, si tenemos un mapa de gran tamaño, nos encontraremos con un gasto de memoria del sistema muy alto y un procesamiento elevado, sin embargo obtendremos un mayor campo de datos en el que apoyarnos. Otro aspecto importante de este parámetro es que la integridad de los datos que contiene debe de ser alta. En este aspecto nos referimos a que si evaluamos una nueva entrada con nuestra implementación del algoritmo, basándonos en un mapa knn con entradas mal etiquetadas, tendremos que la nueva muestra

tendrá unas altas probabilidades de ser mal catalogada. Además tendríamos una alta degradación del sistema, ya que cada nueva muestra erróneamente etiquetada sería añadida al sistema[10] y tendríamos propagación de nuestros errores. Tomaremos como parámetro de la simulación la longitud máxima del mapa[11].

Una vez comentados los parámetros del sistema, solo nos resta hacer una aclaración que hemos tenido en cuenta en nuestra implementación del algoritmo. Tal y como hemos comentado en el capítulo anterior, sabemos que para K>1, podemos encontrarnos problemas de equiprobabilidad entre clases, es decir, que tengamos que dos o más clases tienen K vecinos a la vez a la misma distancia, o lo que es lo mismo, que nuestra muestra de entrada tiene igual de cerca muestras catalogadas de varias clases en la iteración en busca del vecino K-esimo. En este caso hemos tenido en cuenta la sugerencia de los autores del documento que estamos usando como referencia, y para ello consideramos en este caso, como método discriminante la suma de las distancias de los K vecinos de cada clase implicada con respecto a nuestra muestra de entrada. La etiqueta de nuestra muestra será la correspondiente a la clase que tenga una menor distancia en la suma de la 'K' muestras correspondientes.

Con respecto a la implementación base del algoritmo, hemos comprobado su funcionalidad con respecto a la información ofrecida por los autores en el documento de referencia. No la mostramos, ya que nos desviaría del verdadero objetivo del proyecto, no obstante consideramos como justificación del correcto funcionamiento, los resultados mostrados en la comparativa.

Como entornos del algoritmo, volvemos a tener los comentados en el punto anterior con KRLS, obteniendo las mismas gráficas de resultado, para poder realizar una comparación más efectiva entre los algoritmos. Los resultados de las simulaciones los veremos en el epigrafe de comparativa.

[10] En un esquema de entrenamiento por decisión.
[11] El mapa tiene un enventanado, que guarda las 'm' últimas muestras, ya que de no ser así la memoria utilizada por el programa sería demasiado elevada en ejecuciones con gran número de muestras. Además de esta forma evitamos que entren en el entrenamiento muestras catalogadas antiguas que puedan ser erroneas en el estado actual de un entorno dinámico.

## *5.4 Redes RBF*

Al igual que hemos hecho en los algoritmos anteriores, vamos a comenzar este apartado utilizando el documento de referencia [M. Nicolae, C. Botoca, G. Budura; 2004] y nuestra implementación del sistema para realizar un recorrido por los parámetros que rigen el funcionamiento del algoritmo.

Nuestra implementación consta de una red neuronal RBF con '$n_i$' neuronas en la capa de entrada, '$n_h$' neuronas en la capa oculta, con funciones $\phi_i$ con centros '$c_i$' y varianza $\sigma$, y una etapa de salida con una sola neurona que produce un sumatorio del producto de todas las funciones de las neuronas ocultas por sus pesos correspondientes. Una vez comentado este marco general de nuestra construcción, podemos recorrer los parámetros que van a gobernar el funcionamiento de nuestras simulaciones.

- $\alpha$: Son los pesos que se usan en un algoritmo LMS utilizado para actualizar los coeficientes que se multiplican con la salida de la etapa oculta para entrar en la etapa de salida. Si tenemos un mayor valor de $\alpha$ nos encontraremos con que daremos una mayor importancia al error de estimación en el proceso de actualización de $\omega$.
- $\omega$: Estos son los pesos de entrada a la etapa de salida que se actualizan con los parámetros $\alpha$. Estos coeficientes se actualizan con cada entrada al sistema.
- $\rho$: Se define como $\rho = 2 \cdot \sigma^2$, con lo que se entiende como el doble de la varianza de un igualador bayesiano. Este parámetro es empleado en la función gaussiana de las neuronas ocultas. Si tenemos una dispersión mayor, nos encontraremos con que daremos mayor importancia a la diferencia entre la entrada al sistema y el centro de cada neurona de la etapa oculta.
- $c_i$: Son los centros de las neuronas de la etapa oculta. El número de centros determina el correcto funcionamiento del sistema. Si tenemos menos neuronas en la etapa oculta que estados posibles en la etapa de salida, no realizaremos la estimación de manera adecuada, y el sistema oscilará. Por el contrario, si tenemos el

mismo número de neuronas o mayor que de estados de salida y tenemos que sus centros coinciden exactamente con los centros de las clases de salida, tendremos que la convergencia del algoritmo será más rápida.

- $\eta$: Este parámetro aparece en el algoritmo de aprendizaje competitivo al que se someten los centros de las neuronas ocultas. En particular se encarga de premiar al centro de la neurona ganadora mediante un algoritmo LMS.
- $\gamma$: Al igual que $\eta$, también es empleado en el algoritmo de aprendizaje competitivo, pero en este caso, en el algoritmo LMS castigando a la segunda neurona ganadora. De esta forma se busca que el algoritmo encuentre los estados de salida de manera automática y sin datos previos.

El igual que hemos notado en el algoritmo basado en el clasificador KNN, en esta ocasión tampoco vamos a demostrar el correcto funcionamiento de la implementación del algoritmo base, si no que nos sirve[12] como muestra los resultados obtenidos directamente en el entorno de igualación de canal no lineal que aparece en la comparativa de algoritmos.

Una vez comentados todos los parámetros que entran en juego en la ejecución del algoritmo basado en redes neuronales RBF, sólo nos resta ver los resultados obtenidos que aparecen en el epígrafe de comparativa, al igual que hemos realizado con el resto de algoritmos comentados en el presente proyecto fin de carrera.

[12] Aunque no mostramos los resultados de la comprobación de la implementación base del algoritmo, han sido realizados acorde a los resultados mostrados por los autores en el documento de referencia.

## *5.5 Series de Volterra*

Como en los algoritmos anteriores, vamos a comenzar detallando el significado de cada parámetro que rije el funcionamiento de nuestra implementación del algoritmo apoyándonos en el documento de referencia [C.-H. Tseng, E.J. Powers; 1993].

Para nuestra implementación debemos mencionar que empleamos un algoritmo LMS para poder encontrar el valor de los coeficientes de los kernel de Volterra. Este paso no se encuentra implícitamente detallado en el documento de referencia, pero es necesario para el desarrollo del propio algoritmo[13].

Los parámetros que nos encontramos son los siguientes:

- $\beta$: Este parámetro concede una mayor o menor importancia al error existente entre $y_k$, y nuestra estimación obtenida a través del algoritmo LMS. De esta manera, tenemos que garantizar que $\beta$ cumpla las propiedades necesarias, para que la transformación T() sea un mapeo de contracción, garantizando de esa manera la existencia de un punto fijo necesario para tener una estimación correcta.
- $\mu$: Este parámetro es el que controla nuestro algoritmo LMS para obtener los kernels de Volterra del canal empleado. Nuestra entrada al algoritmo serán las estimaciones realizadas y nuestras salidas las entradas al igualador que estamos construyendo. De esta manera simulamos el comportamiento del canal que recorren los datos generados por la fuente.
- Número de bloques: Corresponden al número de elementos en cascada que realizarán la transformación T().

Al igual que para el resto de algoritmos mostrados, emplearemos el mismo entorno de simulaciones comentados para el KRLS (permitiendo de esta manera

[13] Hemos optado por elegir un algoritmo LMS por simplicidad para esa tarea, aunque es posible usar cualquier otro.

realizar una comparación entre las alternativas), siendo los resultados obtenidos los que aparecen en el apartado de comparativa.

En contra de lo que realizamos en el resto de algoritmos incluidos en la comparativa (RBF y KNN), en esta ocasión vamos a entrar en la implementación del algoritmo, tanto en el entorno de igualación estático como dinámico, ya que hemos encontrado problemas que derivan en grandes diferencias con los resultados mostrados por los autores del documento de referencia.

### 5.5.1 Entorno de igualación estático

Realizamos una simulación de igualación estática tal y como comentamos en KRLS, obteniendo unos resultados muy diferentes a los mostrados por los autores en el documento de referencia, por lo que procedemos a realizar un barrido en los parámetros que rijen el comportamiento del algoritmo ($\beta$ y número de bloques del igualador) para determinar el compromiso óptimo que garantice la mínima SER posible. El resultado que obtuvimos bajo una simulación estática de 1000 puntos, y un barrido en $\beta$ de [-2,+2] y de 0 a 40 bloques. El resultado lo podemos ver en la Figura 5.12.

Como se puede ver en la figura, el resultado que nos muestra no es demasiado bueno, ya que el mínimo valor de SER que aparece corresponde con 0.1782 con un valor de $\beta$ de -0.35 y 9 bloques. Este resultado comprobaremos que no es comparable con el obtenido en el resto de algoritmos (eso aparecerá en la comparativa).

Otro punto a tener en cuenta es que sólo existe un pequeño campo de valores que hacen converger al algoritmo. Encontramos que con un valor erroneo de $\beta$ conseguimos saturar al algoritmo, o que no converga a valores con sentido, fuera de la equiprobabilidad. Igualmente tenemos que un valor demasiado alto del número de bloques puede llevar al algoritmo a la inestabilidad. La gráfica que mostramos, nos ayuda a obtener una primera conclusión respecto al algoritmo de volterra, que no es otra que la de que es muy difícil de ajustar. Además una vez elegidos correctamente los parámetros, el resultado que ofrece no es demasiado prometedor en nuestro entorno de igualación no lineal.

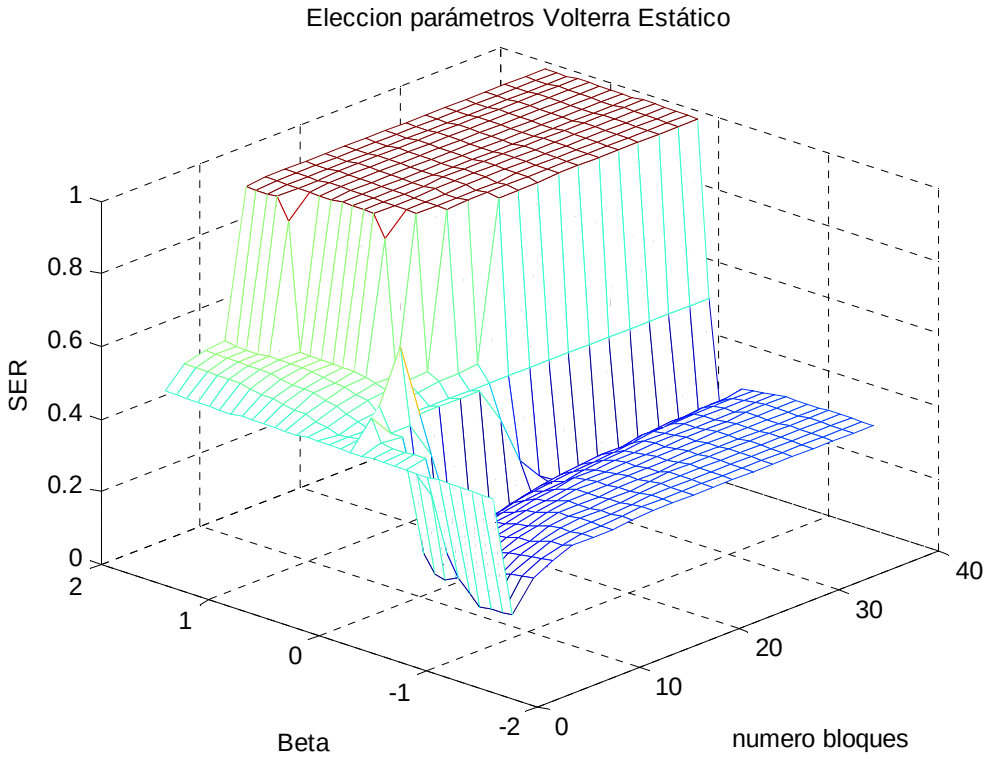



**Figura 5.12 – Barrido de parámetros en entrenamiento estático para algoritmo de Volterra.**

### 5.5.2 Entorno de igualación dinámico

Una vez comentado el entorno estático, podemos mostrar igualmente un ejemplo de entorno dinámico, en el cual hemos realizado una variación senoidal del 2% en los coeficientes del canal[14] para una simulación de 1000 muestras. El resultado del barrido de parámetros realizado, aparece en la Figura 5.13.

A primera vista, podemos ver que dicho resultado es muy parecido al presente para el caso de igualación estática, con la diferencia principal que la franja de equiprobabilidad se ha reducido considerablemente, aumentando en contra la inestabilidad del algoritmo en función del valor de sus parámetros.

En este caso de variación en particular, vemos que el valor óptimo de $\beta$ se ha modificado con respecto al entorno estático, ya que en este caso es muy cercano a 0. Por otro lado, tenemos que el número de bloques si se ha mantenido

[14] Esta señal de variación corresponde con la variación senoidal suave que se utilizará en el apartado de comparativa.

constante en un valor de 9, al igual que la SER mínima conseguida, que queda en torno a 0.2, resultando un valor demasiado pobre igualmente.

Otra conclusión que podemos sacar de este resultado, es aparte del difícil ajuste del algoritmo, que la identificación del valor correcto de los parámetros se encuentra determinado directamente (de manera práctica) con la señal de variación empleada en entornos dinámicos. Esto hace que junto con la deficiente tasa de error, tengamos un algoritmo claramente debil, frente al resto de algoritmos bajo estudio. Este punto lo confirmaremos con nuestros resultados en el apartado de comparativa.

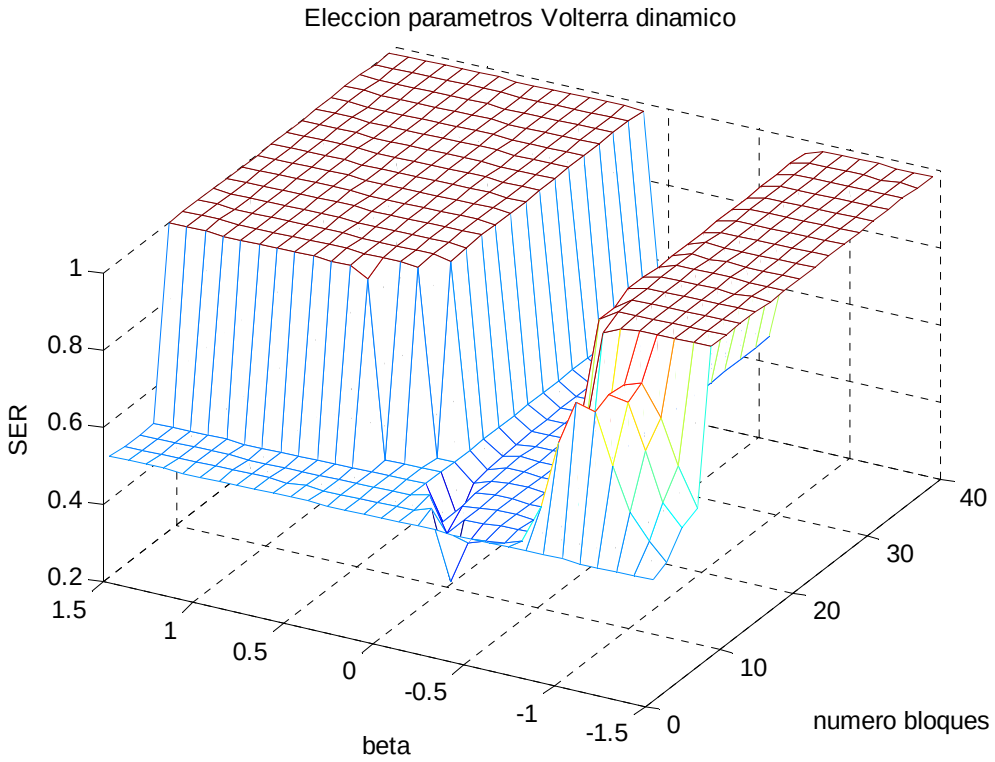



**Figura 5.13 - Barrido de parámetros en entrenamiento dinámico para algoritmo de Volterra.**

## *5.6 Comparativa*

Una vez mostrados los parámetros que rijen el comportamiento de los algoritmos bajo estudio, junto con los resultados básicos de KRLS, vamos a realizar una comparación final que nos guiará hacia las conclusiones finales del presente proyecto fin de carrera.

Excepcionalmente hemos comentado brevemente los resultados básicos del algoritmo de Volterra con lo que pretendemos justificar que la diferencia de entorno de ejecución con el de los autores del documento de referencia, nos lleva a obtener unos resultados muy pobres que comprobaremos en el presente apartado. En el caso de Volterra no hemos podido corroborar los resultados mostrados por los autores en nuestra implementación.

La estructura del punto presente se encuentra enfocada de manera similar a los resultados básicos mostrados para el algoritmo KRLS. En primer lugar, hemos de comenzar por el escenario de entrenamiento estático, apoyándonos en la implementación base de cada algoritmo. Posteriormente comprobaremos el comportamiento de los mismos en un entorno dinámico, para finalizar introduciendo el esquema guiado por decisión con canal dinámico y estático.

El objetivo de este punto es el de mostrar las simulaciones de los diferentes algoritmos bajo una misma simulación y un mismo entorno con el fin de poder comparar sus prestaciones en entornos de igualación no lineal de canal. En ese sentido, en este apartado, queremos mostrar los resultados que van a motivar principalmente las conclusiones finales de nuestro proyecto fin de carrera.

### 5.6.1 Esquema de igualación de canal basado en entrenamiento sobre canal estático

Para comenzar a mostrar los resultados de las simulaciones bajo un entorno estático de igualación de canal, vamos a comenzar detallando los parámetros escogidos para cada algoritmo en la simulación. Estos se pueden ver en la Tabla 5.5.

| | Parámetro | Valor | Observación |
|---|---|---|---|
| **KRLS** | $\nu$ | 0.01 | |
| | $\lambda$ | 0.1 | |
| | $k$ | Gaussiano | Optamos por este kernel frente a los comentados anteriormente, porque con el que mejores prestaciones hemos obtenido en los resultados previos |
| | $a$ | 1 | Valor por defecto |
| **KNN** | K | 3 | |
| | Mapa Knn | 200 elementos | Limitamos el tamaño del mapa Knn, ya que sino se dispararía el tiempo de procesamiento, así como la propagación de errores. |
| **RBF** | $\alpha$ | 0.01 | |
| | $\omega$ | $\underline{0}$ | Vector de ceros de tamaño semejante al número de centros de la capa oculta de la red neuronal. |
| | $\rho$ | 0.2 | |
| | $c_i$ | 8 puntos | Estos puntos corresponden con las nubes de salida de nuestro canal no lineal y que aparecen en la Figura 5.4. |
| | $\eta$ | 0.09 | |
| | $\gamma$ | 0.03 | Damos menos peso a la penalización al segundo ganador. |
| **Volterra** | $\beta$ | -0.35 | Establecemos los valores obtenidos anteriormente en el barrido de parámetros. |
| | $\mu$ | 0.005 | |
| | Nº bloques | 9 | |

**Tabla 5.5 – Configuración de los algoritmos en las simulaciones sobre canal estático**

Los parámetros mostrados en la Tabla 5.5 para los algoritmos KNN y RBF que no han sido comentados en los epígrafes 5.3 y 5.4, se han obtenido realizando un barrido manual, buscando el funcionamiento optimo de los algoritmos con el objetivo de establecer un compromiso entre eficiencia de ejecución y mínima tasa de error. El resultado de estos barridos aparece en la Tabla 5.5. No se han incluido en el presente proyecto, porque no se considera el objetivo principal del mismo. Con respecto a los parámetros de KRLS y Volterra, se detallan sus elecciones en los puntos 5.2 y 5.5 respectivamente.

En este punto podemos enseñar el comportamiento estático de las simulaciones de los algoritmos. Divideremos nuestros resultados en este apartado en dos grupos de gráficas principalmente: por un lado una gráfica de barrido sobre la relación señal a ruido de la fuente de datos ruidosa y por el otro, una gráfica de la evolución de la tasa de error de símbolo en función del número de patrones introducidos al algoritmo.

Para la ejecución de estas simulaciones, hemos tomado el canal constante que aparece de ejemplo en el documento de referencia de KRLS [Y. Engel, S. Mannor, R. Meir; 2004], que aparece en la expresión 5.6.

$$\underline{h} = \begin{bmatrix} 1 & 0.5 \end{bmatrix}^T \qquad \textbf{5.6}$$

### 5.6.1.1 Gráfica SER frente a SNR

Para este resultado, establecemos constante la duración del entrenamiento con un valor de M = 1000 muestras. Este dato se mantendrá en el resto de simulaciones del proyecto.

Con el fin de asegurar la correcta validez de las gráficas que mostramos en este capítulo, volvemos a remarcar la importancia de garantizar que la estimación de la tasa de error de símbolo debe ser realizada con tramas de test de longitud al menos, una década mayor que la inversa de la menor tasa de error de símbolo conseguida en la gráfica. Por este motivo, hemos tomado como longitud de las secuencias de test $T = 10^6$ muestras.

Con el objetivo de obtener gráficas mas suavizadas, se ha realizado un promedio de Montecarlo sobre los resultados, con un número de iteraciones igual a 5.

La gráfica de error simbolo frente a la relación señal a ruido, puede verse en la Figura 5.14, donde se observa que la ejecución del algoritmo KRLS es semejante en terminos de prestaciones a la simulación de RBF y el algoritmo clasificador KNN.

Con respecto al algoritmo de referencia lineal LMS, otenemos el rendimiento esperado para nuestro canal no lineal, por lo que parece no ser capaz de dividir correctamente las nubes de puntos al calcular la frontera de decisión.

Por último, tenemos el caso del algoritmo de Volterra, el cual tiene graves problemas para obtener una tasa de error de símbolo competente, ya que según

los resultados mostrados no parece converger correctamente hacia la solución no lineal de nuestro canal, acercándose únicamente al utilizar SNR altas.

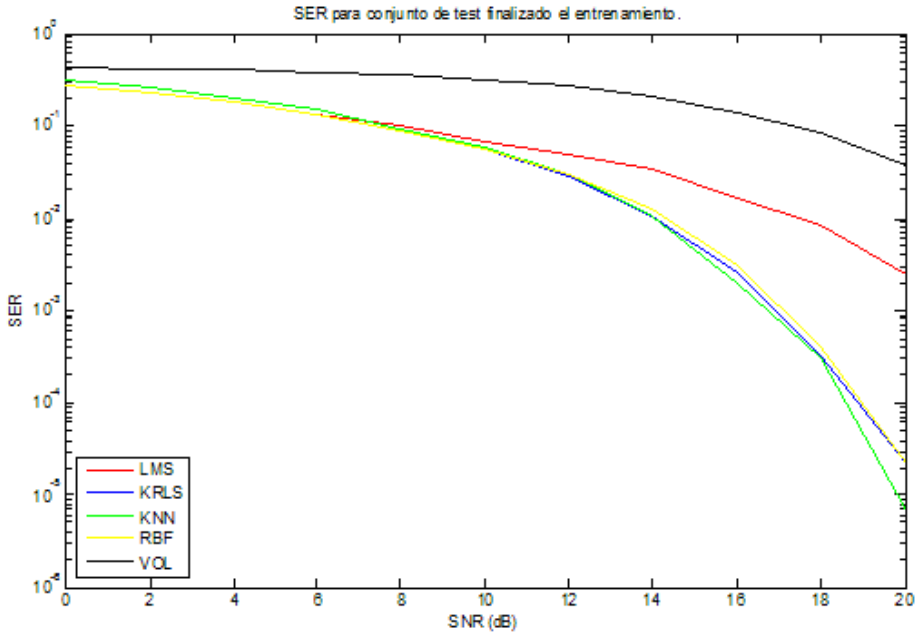



**Figura 5.14 – Gráfica SER frente a SNR para canal estático**

### 5.6.1.2 Gráfica SER frente a patrones de entrenamiento

El segundo resultado que mostramos para el caso del canal estático es el correspondiente al barrido realizado sobre las muestras que empleamos en el entrenamiento del algoritmo. Tal y como hemos comentado para la gráfica de SNR, la longitud del entrenamiento ha sido establecida a M = 1000 muestras, y para la gráfica que aparece en la Figura 5.15 se ha tomado un punto por cada 100 muestras.

Cada punto de la gráfica corresponde con la evaluación de un conjunto de test aleatorio de la misma fuente de datos, con longitud $T = 10^6$, con el objetivo de garantizar la veracidad de los resultados mostrados.

Tal y como podemos observar en la Figura 5.15, el eje de muestras comienza en el primer paso de 100 muestras y finaliza en la muestra 900 (100 muestras antes de la evaluación final del algoritmo)[15]. No obstante en la Tabla 5.6

[15] A fin de mostrar únicamente la variación de la tasa de error de símbolo durante el entrenamiento propiamente dicho, no se ha incluido en la gráfica la estimación final.

podemos ver los resultados de la simulación correspondiente a la secuencia de test aplicada para la muestra de entrenamiento 1000, completando en cualquier caso todos los puntos del entrenamiento. Esta división se ha realizado para diferenciar el resultado final del algoritmo, frente a los intermedios.

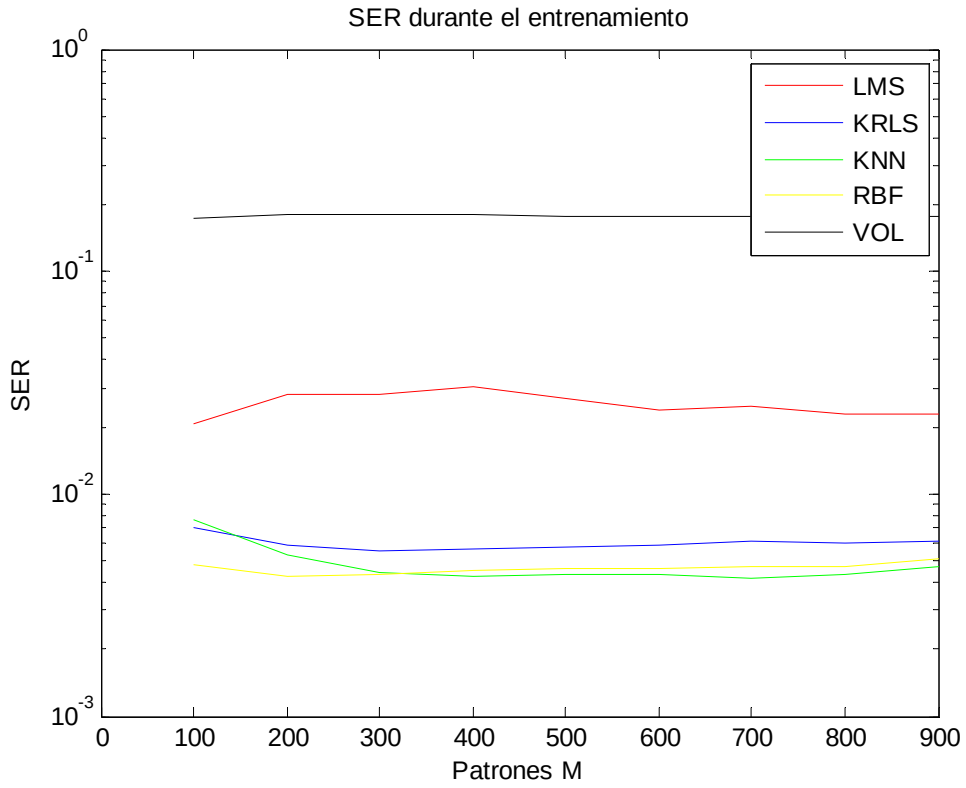



**Figura 5.15 – Gráfica SER frente a patrones para entrenamiento estático**

En la Figura 5.15 podemos comprobar que la estabilidad para el caso de entrenamiento estático se encuentra patente en todos los algoritmos analizados. Se observa un estado estacionario de convergencia de todos los metodos de igualación empleados.

Entrando en particular en cada elemento de la gráfica, podemos determinar que el algoritmo KRLS presenta una mayor tasa de error de símbolo que sus competidores KNN y RBF, siendo a priori la mejor opción durante el proceso de entrenamiento el algoritmo clasificador KNN. Con respecto al algoritmo lineal LMS es evidente que aunque en la gráfica aparece en un estado aproximado de convergencia, no parece la solución más viable en situaciones de entornos no lineales estáticos. Para el algoritmo de Volterra, debemos decir que aunque también se establece en un punto de estabilidad constante, la tasa de error de

símbolo que ofrece es demasiado elevada[16] y no se considera válida, ya que se trata de un algoritmo no lineal que ofrece peores prestaciones que nuestra alternativa lineal, el algoritmo LMS, incluso eligiendo optimamente los parámetros que rigen su comportamiento.

| | KRLS | LMS | RBF | KNN | Volterra |
|---|---|---|---|---|---|
| **SER t = M** | 0.0061 | 0.0218 | 0.0056 | 0.0045 | 0.1752 |

**Tabla 5.6 – Resultados para t = M en entrenamiento estático**

Tal y como hemos comentado anteriormente, en la Tabla 5.6, aparecen los mismos resultados que quedan patentes en la Figura 5.15, donde de nuevo las alternativas mas eficientes, en terminos de SER, son KRLS, RBF y KNN en los mismos ordenes de magnitud.

### 5.6.1.3 Relación de eficiencia para entornos estáticos

Con la finalidad de realizar un estudio de eficiencia entre los tres algoritmos con mínima SER para el entrenamiento estático, podemos observar el tiempo necesario de ejecución de cada uno, para de esa manera poder construir un orden relativo[17] de eficiencia entre ellos. Estos tiempos de simulación para entrenamiento en entorno estático aparecen en la Tabla 5.7 cuyos elementos debemos tener en cuenta que son aproximados, y dependientes de la simulación que se realiza en particular (SNR = 15dB).

| | KRLS | LMS | RBF | KNN | Volterra |
|---|---|---|---|---|---|
| **Tiempo Ejecución [s.] t = M = 1000** | 0.40 | 0.02 | 0.20 | 0.01 | 0.03 |

**Tabla 5.7 – Tiempos de entrenamiento aproximados para M = 1000 muestras bajo entornos estáticos con SNR = 15dB.**

[16] Tal y como hemos comentado y demostrado en puntos anteriores, en el presente proyecto fin no se ha conseguido replicar los resultados del documento de referencia del algoritmo de Volterra, no obstante se ha incluido en todas las gráficas a modo de comparativa.

[17] Cuando nos referimos a relativo, lo hacemos pensando en que los tiempos de ejecución para cada algoritmo son dependientes de la simulación en particular, por lo que el verdadero valor lo obtendremos de la diferencia de tiempos entre las simulaciones de cada uno para un mismo escenario.

| | KRLS | LMS | RBF | KNN | Volterra |
|---|---|---|---|---|---|
| **Tiempo Ejecución [s.] t = M = 1000** | 1.58 | 0.02 | 0.19 | 0.02 | 0.04 |

**Tabla 5.8 – Tiempos de entrenamiento aproximados para M = 1000 muestras bajo entornos estáticos con SNR = 0dB.**

A modo de comparativa con el resto de estudios de eficiencia que realizamos en el presente proyecto fin de carrera, incluimos igualmente los tiempos de ejecución para el caso de SNR de 0dB en la Tabla 5.8. Establecemos este valor de SNR como referencia, ya que es el caso peor de todo el barrido de SNR, al introducir mayor cantidad de ruido en nuestro sistema.

En las tablas podemos ver que entre los tres algoritmos ganadores para el entorno estático, tenemos que el que menor tiempo de entrenamiento oferta, es el algoritmo clasificador KNN, seguido por la red neuronal RBF y por último nuestro algoritmo bajo estudio, KRLS, para ambos valores de SNR mostrados. Con respecto a estos resultados temporales que mostramos, debemos decir que no son del todo comparables, ya que para el caso del algoritmo KNN, como hemos comentado en epígrafes anteriores dentro de este capítulo, el entrenamiento que desarrollamos es únicamente introducir las muestras catalogadas dentro de un array que utilizaremos posteriormente en el proceso de test. En este sentido el proceso de aprendizaje es muy rápido, mientras que en el proceso de testeo pueden incrementarse los tiempos muy rápidamente con el número de muestras incluidas en el mapa knn (este hecho lo podremos observar más adelante). Teniendo en cuenta esta afirmación, vemos que tanto el entrenamiento para RBF como KRLS se encuentran bajo el mismo orden de magnitud[18] (para 15dB) por lo que, al menos en terminos de entrenamiento estático en ese periodo de SNR, los podemos considerar semejantes tanto en terminos de probabilidad de error como de complejidad de ejecución, para las simulaciones que mostramos en el presente proyecto fin de carrera. Por otro lado, si observamos los tiempos para 0dB., esta afirmación deja de ser cierta, ya que el resultado de KRLS incrementa en 1 segundo, mientras que RBF se mantiene en los mismos niveles. Por esa razón debemos considerar a RBF mejor alternativa que KRLS en tiempos de proceso frente a SNR. No obstante, si obviamos el comentario acerca del entrenamiento

[18] Debemos tener en cuenta que los tiempos que se muestran en la Tabla 5.7 se encuentran muy directamente relacionados con los parámetros elegidos para la simulación, por lo que el entorno es comparable únicamente bajo cotas de SER similares, como es el caso del algoritmo KRLS y RBF.

del algoritmo KNN, es evidente que la mejor opción sería el clasificador entre nuestras tres mejores alternativas.

#### 5.6.1.4 Gráfica de contorno para entornos estáticos

Este resultado muestra las regiones de decisión que generan los algoritmos bajo estudio. Con esta gráfica esperamos poder apoyar los resultados mostrados en las gráficas de SER de epígrafes anteriores. Sólo mostraremos esta gráfica para el caso estático, ya que únicamente nos sirve para establecer la relación visual entre la tasa de error de símbolo y la frontera de decisión que divide las nubes de puntos generadas tras pasar las muestras de nuestra fuente a través de un canal no lineal. En el caso dinámico sería muy complicado realizarlas, ya que las nubes de puntos generadas a la salida del canal serían variables.

Si la frontera de decisión separa las nubes de puntos de manera más limpia, tendremos una tasa de error menor, que si la frontera divide los grupos de una misma clase.

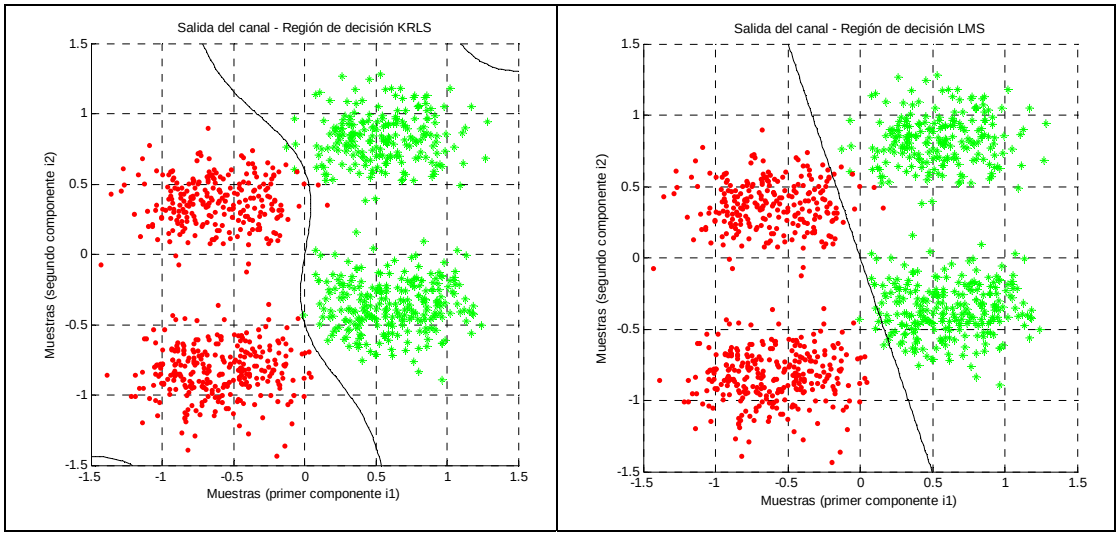
En la Figura 5.16 podemos ver las lineas de contorno para nuestros algoritmos bajo estudio. Se ha realizado con una simulación de entrenamiento con $M = 10^5$ muestras[19] previamente catalogadas en las dos clases presentes en nuestra fuente aleatoria de símbolos BPSK. Los puntos rojos corresponden con un valor ‘1’ de la entrada frente al ‘-1’ para los asteriscos verdes. Posteriormente se realiza un barrido bajo todos los puntos de la representación (entre -1.5 y 1.5) siendo la componente horizontal la primera componente del agrupamiento de las muestras y la vertical la segunda.

Examinando los resultados que se muestran en la Figura 5.16 podemos ver a primera vista que el algoritmo LMS construye una frontera de decisión recta que para el caso de canal no lineal que tenemos bajo estudio hace que las nubes de puntos de ambas clases no se separen convenientemente. Esto justifica que su tasa de error de símbolo sea mayor que los algoritmos KRLS, KNN o RBF. Por otro lado, tenemos la frontera de decisión de Volterra, que demuestra convenientemente la tasa de SER tan elevada que hemos comprobado en gráficas anteriores. Genera una frontera de decisión no lineal, sin embargo parece estar

[19] Corresponden con los puntos que se muestran en la gráfica.

desplazada con simetría par, separando correctamente solo la mitad de los puntos de salida de nuestro canal. De esta forma se puede corroborar las cotas de error conseguidas cercanas a 0.5. Este resultado es otro pilar para demostrar que este algoritmo no funciona correctamente en nuestras simulaciones.

Entrando en detalle en los algoritmos con mejores resultados, vamos a comenzar por el que nos ocupa en el presente documento. El algoritmo KRLS presenta una frontera no lineal que parece separar correctamente las nubes de puntos que representan la salida de nuestro sistema. Esto certifica las bajas tasas de error de símbolo que ofrece este algoritmo. No obstante aparecen dos nuevas fronteras de decisión en los límites (1.5,1.5) y (-1.5,-1.5) que delimitan de manera precisa las dos clases, pero que pueden llevar a error al algoritmo en presencia de una desviación de la fuente mayor o de datos atípicos o cambios de canal[20]. Con respecto al algoritmo RBF, notamos que la frontera de decisión que genera es no lineal, pero de un grado menor al de KRLS. Dicha frontera, al no tener un grado tan elevado provoca que el algoritmo incurra en fallos ante salidas del canal muy cercanas entre sí y difíciles de categorizar. Por último tenemos el resultado del algoritmo KNN, que muestra claramente una frontera muy irregular característica de algoritmos basados en agrupamientos. No obstante aunque dicha frontera parece azarosa, se puede comprobar que el contorno de dicha tendencia es muy parecido a la solución que ofrece el algoritmo KRLS bajo estudio a excepción de las nuevas barreras que presenta éste.




[20] Hay que tener en cuenta que las gráficas que mostramos corresponden con el entorno de entrenamiento estático.

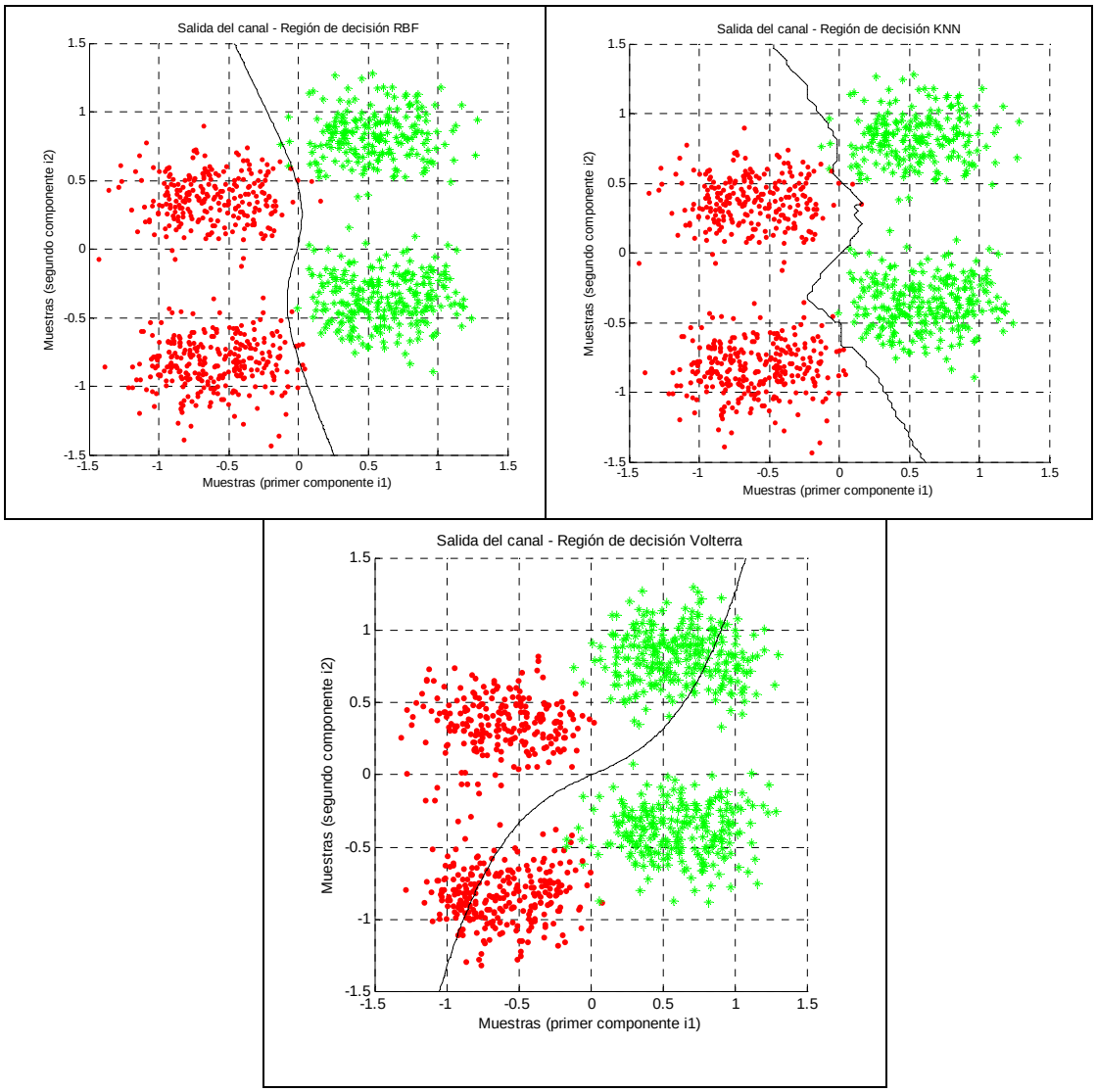



**Figura 5.16 –Gráficas de contorno para entornos estáticos**

### 5.6.1.5 Gráfica de error para entornos estáticos

Para terminar de caracterizar el comportamiento estático de nuestros algoritmos vamos a presentar los resultados que hemos obtenido en terminos de error cuadrático para cada uno de ellos. Dichos resultados se pueden ver en la Figura 5.17.

En dicha gráfica no aparece el resultado del algoritmo KNN, ya que al basarse en una ventana de muestras de entrada etiquetadas sobre las cuales calculamos distancias, tenemos que el error del algoritmo sólo pueden ser

muestras con valores $\pm 1$(dependiendo del valor de la etiqueta obtenida para la nueva muestra), lo cual no tiene sentido con las gráficas de error cuadrático medio que mostramos.

En la gráfica vemos como los cuatro algoritmos que mostramos han alcanzado la convergencia sobre el entorno estático donde se realiza el entrenamiento, manteniendo una cota de error constante.

Observamos que el algoritmo KRLS ofrece una tasa de convergencia muy elevada (en torno a 100 muestras), lo cual le lleva a tener el tiempo de convergencia más reducido de todas las alternativas. El siguiente algoritmo que presenta buena velocidad de convergencia es Volterra, ya que como se puede ver, dura aproximadamente el mismo número de muestras que KRLS, con la ventaja de que su error es significativamente menor[21].

El algoritmo RBF, tiene una velocidad de convergencia algo mayor, en torno a 400 muestras, teniendo además una varianza de ruido muy superior a KRLS y Volterra. Además presenta una variabilidad de ruido muy elevada.

Por último, la mayor tasa de variación, pertenece al algoritmo LMS, lo cual era de esperar, por su naturaleza lineal. A la vista de los resultados necesita mayor número de muestras para alcanzar una solución adecuada en entornos no lineales. La convergencia se consigue en torno a las 900 muestras, con una varianza de ruido importante, aunque menor que el algoritmo RBF, y una media de error elevada. Se notan claramente los problemas que tiene el algoritmo para alcanzar un estado estable, debido a que no consigue engancharse a la solución no lineal.

[21] El error mostrado para el algoritmo de Volterra corresponde con el presente en el algoritmo LMS que utilizamos en su implementación para estimar los kernels de Volterra del equivalente del canal.

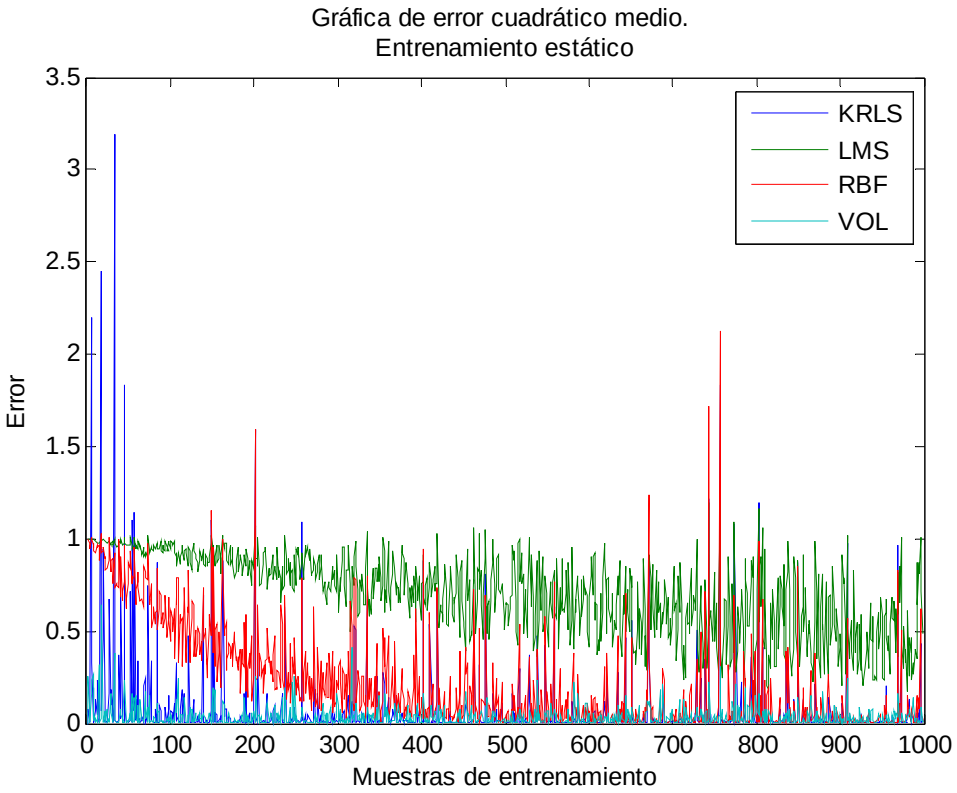



**Figura 5.17 – Gráficas de error para entornos estáticos**

### 5.6.2 Esquema de igualación de canal basado en entrenamiento sobre canal dinámico

Una vez comentados los resultados bajo el entorno de entrenamiento estático, podemos complicar levemente nuestros experimentos, a fin de adaptar nuestro sistema a un esquema con canal dinámico variante con el tiempo. De esta manera proponemos un nuevo entorno, tal y como se especificó en el punto 5.2.3 del presente proyecto.

Con el objetivo de unificar el estado inicial de estas simulaciones, se toma un entrenamiento estático previo a la variación de canal con el fin de garantizar un estado estacionario similar para todos los algoritmos bajo estudio. En particular tendremos una longitud de entrenamiento estático de M = 1000 muestras, al igual que en los resultados mostrados en el esquema de igualación estático.

Una vez finalizada esa etapa estática[22], se continúa con el entrenamiento dinámico, en el que realizaremos la variación de canal basándonos en señales básicas. De esta manera procederemos al estudio de convergencia y de SER para nuestros algoritmos, al introducir una senoide suave, senoide media, escalón suave y escalón medio en los coeficientes de canal. En la Tabla 5.9 podemos ver más en detalle los canales variantes empleados en este apartado.

En las gráficas de la tabla aparecen dos trazos, correspondientes a los coeficientes normalizados del canal empleado como base estática (Expresión 5.6). Partiendo de estos coeficientes producimos diferentes variaciones[23]

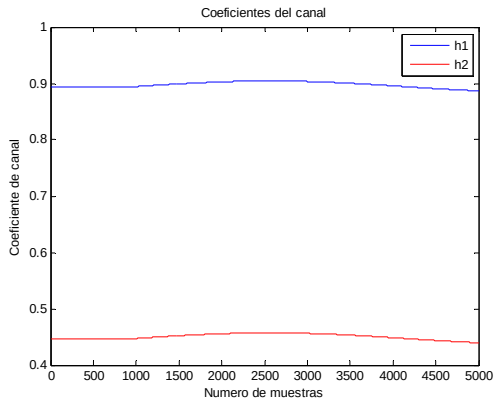
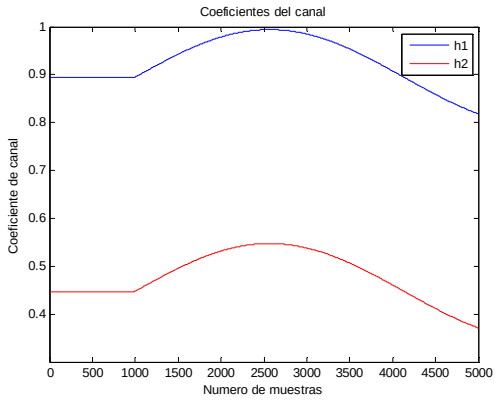
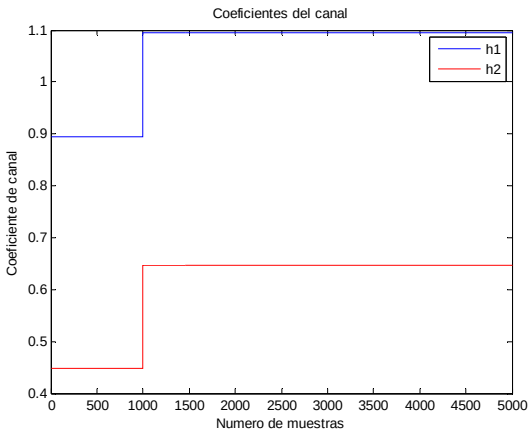
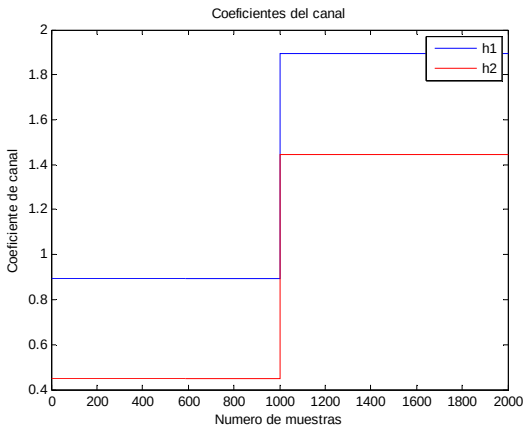
| | | Variación | |
|---|---|---|---|
| | | Suave | Media |
| Tipo | Senoidal |  |  |
| | Escalón |  |  |

**Tabla 5.9 – Canales empleados en entrenamiento dinámico**

[22] Este punto ofrece los mismos resultados en terminos de SER que se muestran en el epígrafe de entrenamiento estático. Por esa razón en este epigrafe no se incluiran de nuevo. Nos centraremos únicamente en los resultados dinámicos.

[23] Estas variaciones son las mismas para ambos coeficientes.

Como se puede ver en la Tabla 5.9, hemos empleado dos canales básicos empleados en comunicaciones como son la señal senoidal y el escalón.

- Señal senoidal: Se ha optado por una señal con una pulsación:

$$\omega = 1\times10^{-3} rad / seg$$

Que corresponde con un periodo de 1570 muestras aproximadamente. Utilizamos dos variantes de señales catalogadas a efectos prácticos como de variación suave y media. Para el primer caso tenemos una variación máxima de $1\times10^{-2}$ sobre cada valor estático de los coeficiente de canal, mientras que para el segundo tenemos una variación de $1\times10^{-1}$. En terminos relativos, podemos determinar que la variación en terminos porcentuales que introduce nuestra señal senoidal en sus niveles máximos corresponde con un 2% para la señal suave y un 22% para la señal media aproximadamente[24]. Podemos comprobar la variación que supone en cada componente cada una de las señales en la Tabla 5.10.

Estos dos valores se han tomado como ejemplos para demostrar el comportamiento de los algoritmos ante este tipo de variaciones.

- Señal escalón: Para esta señal básica, también catalogamos dos niveles, suave y medio: El nivel suave supone un incremento sobre el valor del coeficiente de canal de $2\times10^{-1}$, mientras que el nivel medio corresponde con un incremento en 1. Esto se traduce en una variación relativa en términos máximos para el primer caso de 44% y para el segundo de 222% aproximadamente, en relación al valor menor estático de los coeficientes de canal, tal y como aparece en la Tabla 5.10.

[24] Para la elección de estos valores se ha tomado como referencia el valor del coeficiente más reducido, con el fin de determinar la máxima variación relativa entre los dos coeficientes del sistema.

| | Senoidal Suave | Senoidal Media | Escalón Suave | Escalón Medio |
|---|---|---|---|---|
| Coeficiente 1 (0.9) | 1.11% | 11.11% | 22.22% | 111.11% |
| Coeficiente 2 (0.45) | 2.22% | 22.22% | 44.44% | 222.22% |

**Tabla 5.10 – Tasas de variabilidad frente a los coeficientes de canal de las señales de variación dinámica empleadas**

Las variaciones relativas máximas de los canales empleados en los experimentos son muy dispares. Esto es debido principalmente a que con el canal senoidal, queremos medir la capacidad de enganche de nuestros algoritmos. Por esa razón proponemos una variación de un 2% máximo en los coeficientes para la alternativa suave, y una variación de 22% para la alternativa media. La primera señal busca verificar la capacidad de enganche con una transición muy leve, mientras que la segunda tiene como objetivo comprobar esta capacidad con una variación una década mayor. Con respecto al canal escalón, no elegimos las mismas cotas de variación, ya que nuestra finalidad es poner a prueba la capacidad de adaptación a una situación completamente nueva del canal. Por esa razón partimos de una tasa de variación máxima en los coeficientes del canal de 44% inicialmente para la alternativa suave. Con la otra cota que proponemos para la alternativa media, 222%, queremos estudiar el comportamiento extremo de adaptación de nuestros algoritmos.

Debemos notar que todos los canales que se han empleado en los experimentos y que acabamos de comentar, realizan la variación a partir de la muestra 1000, con el objetivo de garantizar un estado estático estacionario en los algoritmos. Teniendo esto en cuenta y uniendolo al hecho de realizar un entrenamiento estático previo al entrenamiento dinámico como comentamos, tendremos un total de 2000 muestras de entrenamiento estático previo a la variación de los coeficientes de nuestro canal. Esta longitud nos garantiza un estado convergente de nuestros algoritmos.

Con respecto al valor de los parámetros de los algoritmos empleados en las simulaciones de este punto, debemos decir que son semejantes a los mostrados para el esquema estático de funcionamiento, por lo que la Tabla 5.5 sigue vigente para este punto.

### 5.6.2.1 Gráfica SER frente a SNR

Para la ejecución de este resultado hemos empleado el esquema comentado en el punto anterior, en el que nos encontramos con un entrenamiento estático con los coeficientes que especificamos en la expresión 5.6 durante 2000 muestras[25]. Posteriormente a este estado de entrenamiento estático, tendremos una variación de canal que durará 3000 muestras dando lugar a nuestros experimentos de 5000 muestras totales.

Tal y como hemos comentado anteriormente, nuestro objetivo es garantizar que las gráficas que mostramos como resultados sean válidas. Para ello todos los resultados obtenidos en la estimación de tasa de error de símbolo se realizan con tramas de test de longitud al menos una década mayor que la inversa de la menor tasa de error de símbolo conseguida en la gráfica. Igualmente que para los experimentos estáticos, se ha elegido una longitud de tramas de test de $T = 10^6$ muestras.

Para los resultados que se muestran en este epígrafe se ha vuelto a realizar un promedio de Montecarlo con 5 iteraciones para suavizar las gráficas.

A continuación mostraremos los resultados para los cuatro canales variantes empleados en los experimentos:

- Canal senoidal suave: El resultado de este canal corresponde con la Figura 5.18. En ella podemos ver que nuestra gráfica es válida con nuestra simulación de test de $10^6$ muestras ya que los resultados llegan únicamente a tasas de $10^{-5}$. En ella vemos que el peor resultado que aparece corresponde con el algoritmo Volterra al igual que pasaba en entrenamientos estáticos. Esto está de acuerdo con lo que comentamos en epígrafes anteriores acerca del comportamiento de este algoritmo en particular. No obstante, Volterra ofrece la misma tasa de error y la misma curva que ofrecía en la alternativa estática que aparece en la Figura 5.14, por lo que podemos determinar que aunque su tasa de error sea muy

[25] En los experimentos de entrenamiento guiado por decisión realizaremos un análisis sobre la convergencia de los algoritmos previo a la variación de los canales. Lo realizamos en ese experimento porque lo consideramos el objetivo final del presente proyecto fin de carrera.

elevada, admite de manera correcta las variaciones senoidales en los niveles que introduce la señal senoidal suave.

Con respecto al resto de algoritmos, tenemos que tener especial atención en las irregularidades que presenta KNN entre los 0 y 8 dB. En este intervalo tiene unas prestaciones por debajo de la alternativa lineal. Esto nos hace pensar que la variación senoidal provoca comportamientos no eficientes sobre este algoritmo. A partir de los 8 dB modifica su tendencia para igualar las prestaciones de KRLS y RBF. Pese a la variación tan reducida que supone este canal sobre los coeficientes lineales de partida, el algoritmo KNN se comporta mal, lo que nos indica que no soporta correctamente las variaciones de canal. Podemos suponer que para mantener cotas eficientes de SER necesita más SNR que el resto de alternativas. Esta hipótesis la podremos corroborar con los siguientes canales. En principio podríamos mejorar los resultados de este algoritmo si aumentamos su ventana de trabajo (tal y como hemos comentado estamos trabajando con una ventana constante para todos los experimentos de 200 muestras), ya que aumentariamos el promedio que se realiza sobre las entradas, haciendo que las variaciones sean suavizadas en el algoritmo. No obstante este cambio supondría un aumento exponencial del procesamiento del algoritmo, lo cual no es deseado en tiempos de ejecución, al desear un algoritmo online como requisito.

Con respecto al resto de algoritmos, podemos determinar que tanto KRLS como RBF siguen ofreciendo la misma tasa de error durante todo el barrido de SNR, y la mínima de todos los algoritmos mostrados, por lo que suponen las mejores alternativas en estos términos. Además soportan bien la variación de canal que supone la señal senoidal suave.

Por otro lado, tenemos el algoritmo LMS, que ofrece un cambio en la tasa de error, sobre todo a partir de 8 dB, lo que le hace llegar a tasas menores a $10^{-3}$ frente a las de $10^{-2}$ que ofrecía en el entorno estático. Esto supone una mejora considerable en sus prestaciones, ya que su complejidad sigue siendo la misma, y la mínima de todos los algoritmos mostrados.

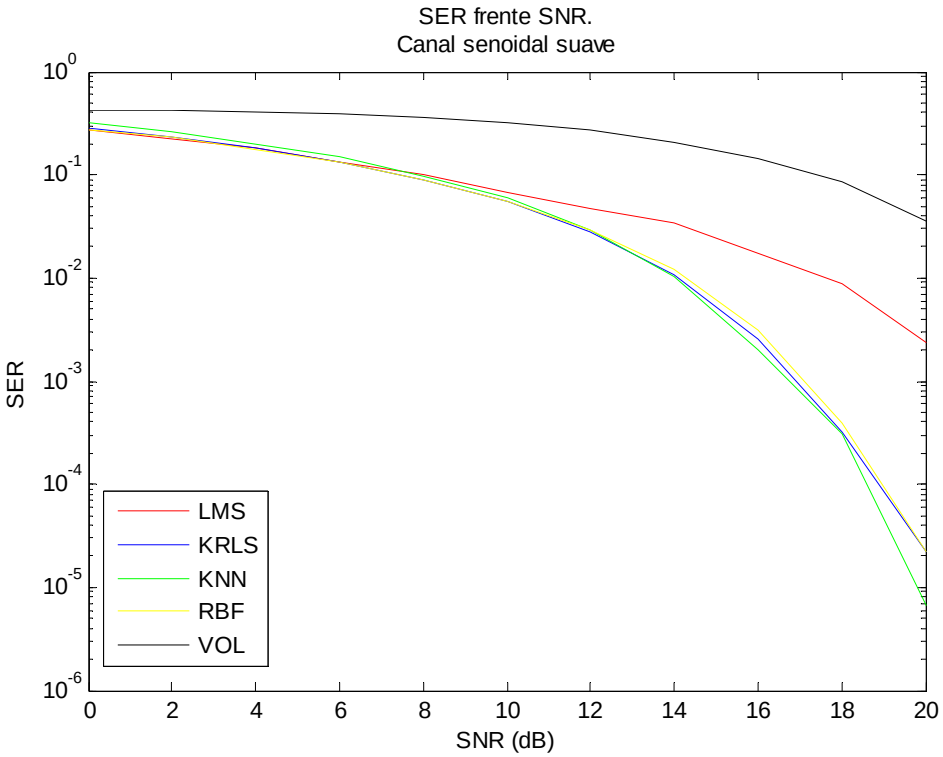



**Figura 5.18 – SER frente a SNR para canal senoidal suave**

- Canal senoidal medio: En la Figura 5.19 vemos el resultado de aplicar la variación de canal senoidal medio. A primera vista determinamos que el comportamiento de Volterra se ha corrompido completamente. En la anterior señal senoidal se había mantenido el comportamiento del algoritmo, sin embargo ahora sólo obtenemos un valor fijo de tasa de error de 0.5 aproximadamente, lo cual nos lleva a pensar que el algoritmo no ha sabido adaptarse a esta nueva señal de variación.

Con respecto a KNN, habiamos dejado patente la inestabilidad que presentaba ante la variación senoidal suave. En este caso nos encontramos con que la situación no se recupera a partir de 8dB como ocurría en la Figura 5.18. En este caso la variación provoca que el algoritmo pierda completamente la estabilidad y la convergencia alcanzada en el proceso estatico previo. Se observa que a partir de 8dB parece intentar recuperarse como lo hacía en el caso anterior, pero en esta ocasión vuelve a corromperse la respuesta ante la nueva variación, para llegar finalmente a cotas de error de 10-1.

Con respecto a los algoritmos KRLS y RBF tenemos que siguen ofreciendo las mismas prestaciones, aunque en este caso la tasa de error que aparece no aumenta debido a la variación que se introduce al sistema. En concreto, la SER resultante es una década mayor que la que se muestra en el entrenamiento estático.

Por último, con respecto al algoritmo lineal LMS, tenemos que aunque la SER que ofrece es mayor que la que aparecía en la variación senoidal suave, en esta ocasión tenemos aproximadamente la misma tasa de error de símbolo y tendencia en la curva que la que aparece en el entrenamiento estático.

Observando los resultados obtenidos en la variación senoidal, podemos determinar que el algoritmo que mejor se ha adaptado a la variación de los coeficientes del canal lineal es el LMS. No obstante ofrece una tasa de error muy alta con respecto a las alternativas de RBF y KRLS. Por otro lado LMS es menos costoso en términos computacionales que los otros dos algoritmos. En conclusión preferiríamos KRLS o RBF en situaciones con variabilidad acotada y reducida en los límites de la onda senoidal media, por su reducida tasa de error. Sin embargo, si la tasa de variabilidad es mayor, la alternativa LMS deberá tenerse en cuenta firmemente, ya que proporciona la mayor estabilidad con menor costo de procesamiento. En ese sentido, tenemos que apuntar igualmente la alternativa de mejorar el comportamiento de KRLS y RBF por elección de sus parámetros, pero no parece aceptable, ya que aumentaría sus tiempos de proceso.

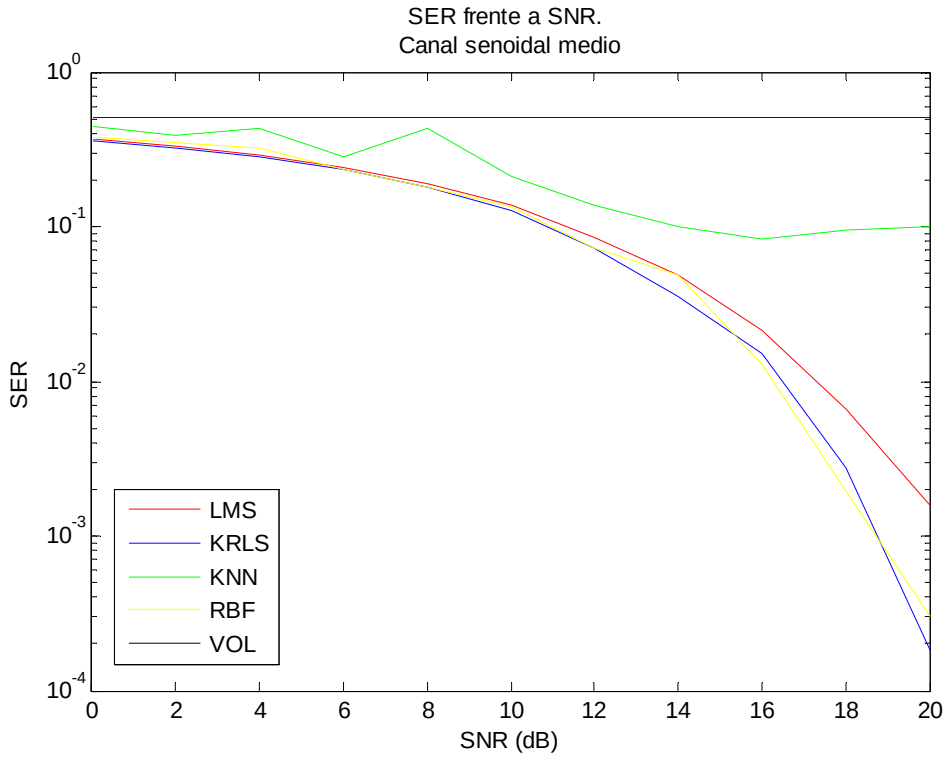



**Figura 5.19 – SER frente a SNR para canal senoidal medio**

- Canal escalón suave: En la Figura 5.20 aparece el resultado de nuestro sistema de canal variante ante una señal escalón suave como la que hemos detallado con anterioridad.

Comenzamos por el algoritmo de Volterra, en el cual se puede ver un empeoramiento del resultado ofrecido en el entrenamiento estático. No obstante se puede determinar que para una tasa de variación mayor que la que supone la señal senoidal media, como es el caso que nos ocupa, obtenemos mejores prestaciones, por lo que podemos deducir que la degradación que aparece en el caso senoidal es debido al seguimiento de la señal, no al enganche de la misma ante una nueva situación. Otra prueba de esta hipótesis es que el resultado de canal senoidal suave es muy parecido al estático.

El siguiente algoritmo es la alternativa lineal LMS. Al igual que ocurría con Volterra, tiene una degradación de la respuesta con respecto a la alternativa estática. En este caso nos encontramos con que la respuesta de LMS es bastante peor que la que ofrece en el canal senoidal medio con unas tasas de variación semejantes. Esto nos lleva a pensar que el

algoritmo LMS es muy sensible a variaciones bruscas del canal, no siendo capaz de adaptarse convenientemente.

La alternativa RBF que tan buenos resultados nos había ofrecido con la variación senoidal, parece no comportarse igualmente ante un canal escalón suave. En esta ocasión sólo llega a cotas de $10^{-3}$ de SER como mínimo frente a los $10^{-5}$ del entrenamiento estático o $10^{-4}$ de variación senoidal. En comparación con el comportamiento de KRLS o KNN podemos comprobar que la variación escalón no produce buenos resultados de enganche a RBF.

Con respecto a KRLS, nos encontramos con que presenta el mejor resultado de todos los algoritmos bajo estudio, al menos en comparativa, ya que en este caso ofrece cotas de $10^{-4}$, sólo una década mayor que las ofrecidas en el caso estático o senoidal suave, e igual que en el caso senoidal medio. Por esto último parece corroborarse que su capacidad de enganche y seguimiento es de las mejores de las alternativas que mostramos.

Por último nos queda detallar el resultado de algoritmo KNN, que en este caso nos ofrece el mejor resultado en SER frente a SNR, llegando a valores de $10^{-5}$ en 20 dB. No obstante, aunque este resultado parece muy esperanzador, no debemos olvidar el comportamiento que presentó el algoritmo ante una señal senoidal de una variación semejante a la del caso actual. Esto nos lleva a pensar, que tiene gran capacidad de enganche, pero no de adaptación y seguimiento.

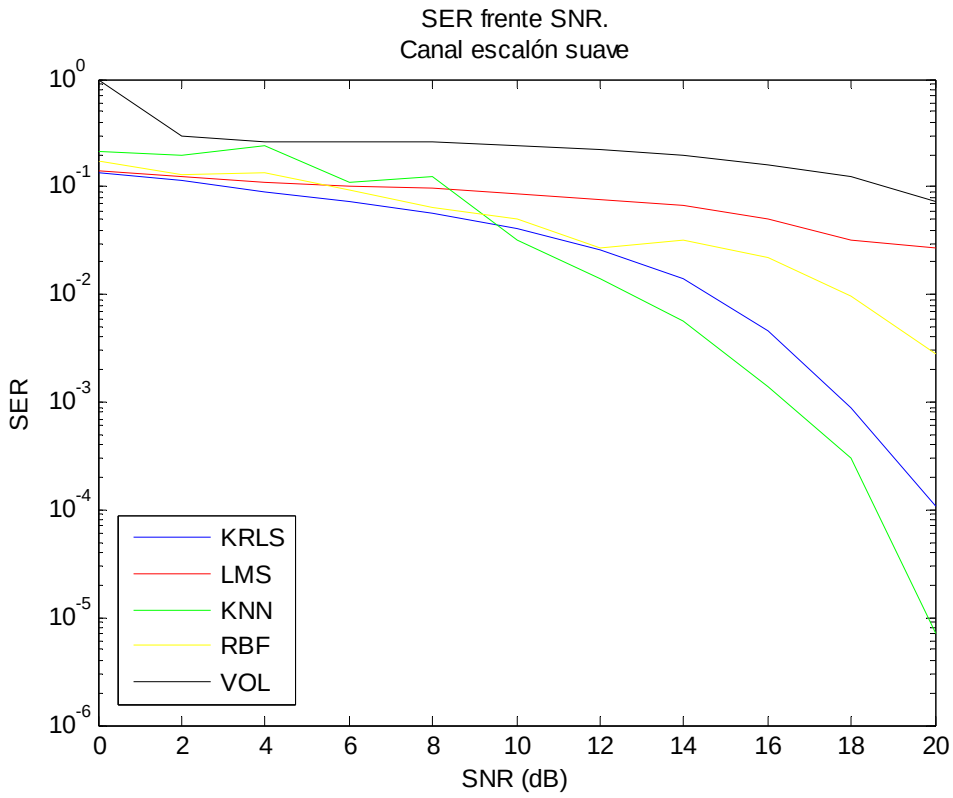



**Figura 5.20 –SER frente a SNR para canal escalón suave**

Debemos detallar un problema que hemos encontrado durante la simulación de la gráfica que nos ocupa. Observamos una duración elevada en la ejecución del entorno dinámico, principalmente a partir de la muestra 2000, que corresponde con el punto exacto de variación de la señal escalón.Para dar una explicación de este comportamiento, podemos ver la Figura 5.21 en la que se muestran las inserciones que se realizan en el diccionario del algoritmo KRLS durante la variación. Como podemos ver, aparece un comportamiento exponencial sobre los elementos del diccionario (cuantos más elementos existan en el diccionario, mas tiempo de procesamiento necesitará la ejecución del algoritmo KRLS). En el caso que nos ocupa, consideramos que la cota final de la exponencial, en torno a 200 muestras en el diccionario, no ralentizaban demasiado, por lo que se mantuvo la configuración de ejecución sin cambios.

Se observa igualmente en la Figura 5.21, un escalon previo entre la muestra 1000 y 2000. En ese sentido, debemos mencionar que la gráfica que se muestra corresponde con el entrenamiento dinámico únicamente, por lo que los datos comienzan desde la muestra 1000. Las muestras anteriores son entrenamiento estático y no aparecen con datos en la figura.

Igualmente podemos observar que el valor constante de inserciones en el diccionario que aparece entre las muestras 1000 y 2000 se corresponde con el valor estático de los coeficientes de canal de la señal de variación de canal.

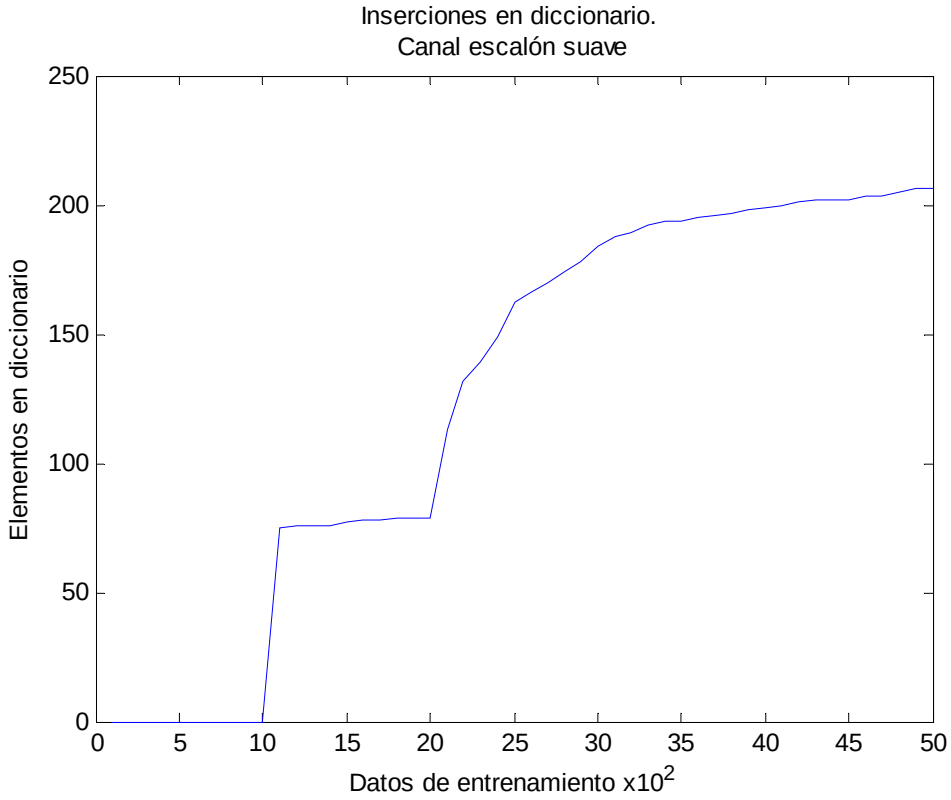



**Figura 5.21 – Problemas de inserciones en KRLS para canal escalón suave**

- Canal escalón medio: En la Figura 5.22 aparece el resultado de la simulación con una señal escalón con una tasa de variación máxima sobre los coeficientes de canal de 222%. Como se puede entender, esta variación es muy elevada, y nos lleva a proponer un estado extremo a los algoritmos bajo estudio, con el objetivo de comprobar sus prestaciones.

Realizando una visión general, de los resultados obtenidos, podemos notar que la elevada tasa de variación ha provocado una degradación importante en la respuesta de todos los algoritmos.

El peor resultado que mostramos es para el algoritmo de Volterra, el cual ofrece una tasa de error de 1. Esta tasa tiene su justificación en que el algoritmo no converge, por lo que ningún valor es considerado válido (si

convergiera, pero los resultados no fueran correctos, tendría sentido un resultado de 0.5 en la SER).

Con respecto a RBF comprobamos que el resultado está completamente degradado, con respecto al escalón suave. Esto no hace más que confirmar mediante este caso extremo, que el comportamiento del algoritmo RBF empeora gravemente cuanto mayor es el salto de la señal de variación.

El algoritmo LMS ofrece un resultado similar al mostrado en la señal de variación escalón suave, con cotas del orden de $10^{-1}$. Si bien la tasa de error que aparece no es demasiado relevante, debemos tener en cuenta que la variación de los coeficientes lineales del canal es muy elevada, y que el comportamiento del algoritmo no se ha modificado, lo cual muestra una determinada robustez.

Con respecto al algoritmo bajo estudio KRLS, tenemos que su comportamiento se ha degradado completamente con respecto al resultado mostrado en la señal escalón suave. En esta ocasión tenemos cotas de 0.5, lo cual quiere decir que no se están obteniendo resultados correctos, y que el algoritmo no se encuentra enganchado con la nueva señal (no ha modificado su frontera de decisión).

A modo de detallar de una manera más efectiva el comportamiento del algoritmo KRLS, podemos ver la Figura 5.23, en la que se muestran las inserciones en el diccionario del algoritmo durante el periodo de la simulación. El formato de gráfica es semejante a la mostrada en el caso anterior en la Figura 5.21. Comparando ambas gráficas, vemos que la exponencial de inserciones frente a muestras tiene unos límites mucho mayores, llegando incluso a no encontrar cota máxima en las 5000 muestras de la simulación. Esto nos dice que el algoritmo no tiene suficiente información en su diccionario para poder representar todas las nuevas muestras de entrada que van llegando, por lo que tiene que ir incluyendolas en el diccionario. Si bien, el porcentaje de reducción del algoritmo sigue siendo bajo (un 10% de las muestras totales se encuentran en el diccionario), no parece demasiado eficiente a efectos de computación. Por este motivo, hemos experimentado unos tiempos de

simulación muy elevados en este sentido. Esto nos ha llevado a limitar el número de muestras de la simulación de las 5000 iniciales a 2250 (se eligió este valor por un compromiso entre tiempo de computación y elementos del diccionario) con el objetivo de acortar la simulación a tiempos realizables, garantizando en cualquier caso, que el cambio de canal fuera aplicable. En estas condiciones no dejamos al algoritmo KRLS adaptarse a la nueva situación. No obstante el problema se encuentra en que no olvida estados anteriores del canal, sino que los sigue guardando en su diccionario. Esto es muy poco eficiente en entornos dinámicos, ya que ralentizamos la ejecución por datos que no tienen sentido en el instante actual.

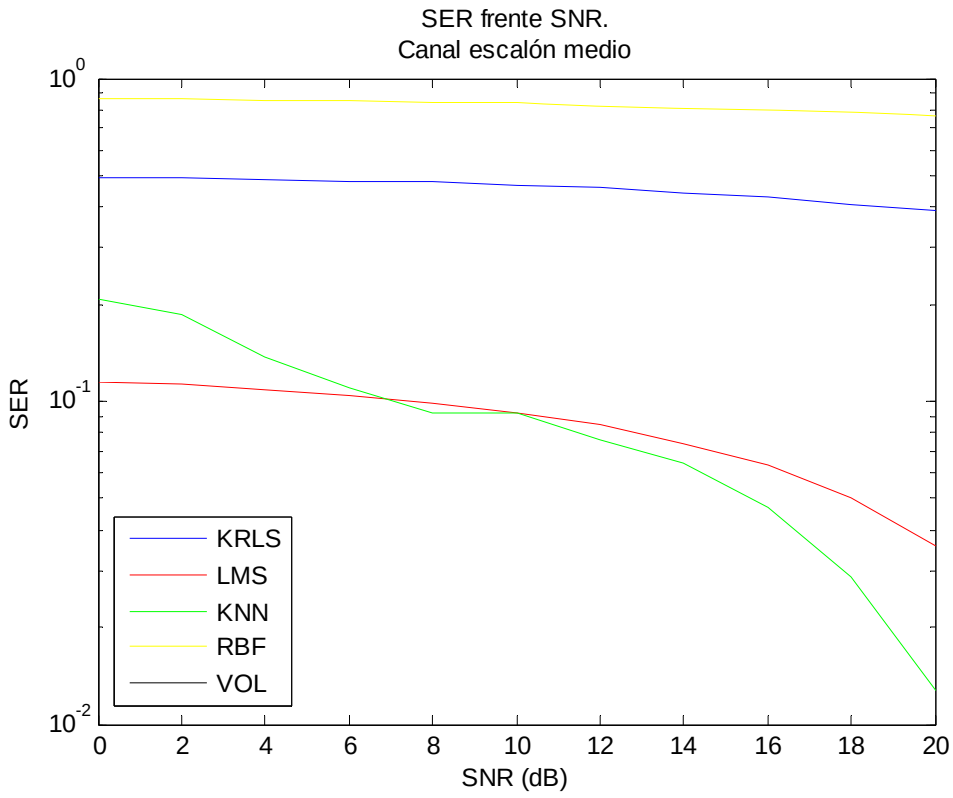



**Figura 5.22 - SER frente a SNR para canal escalón medio**

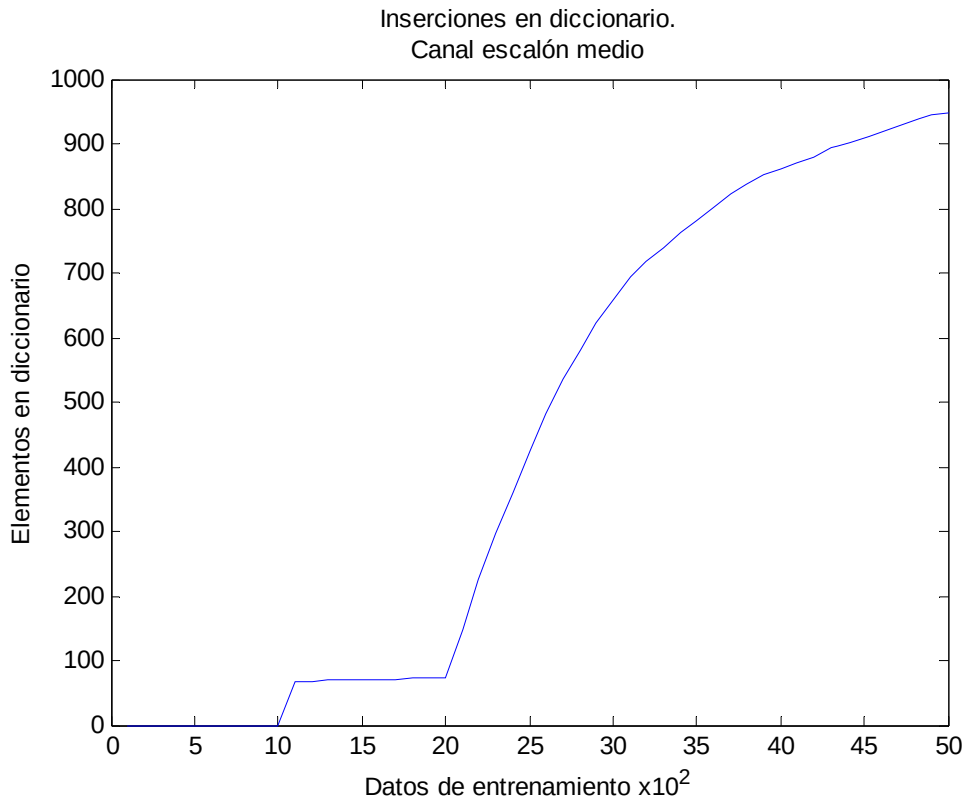



**Figura 5.23 - Problemas de inserciones en KRLS para canal escalón medio**

Nos queda por último comentar el comportamiento del algoritmo KNN, el cual ofrece la mejor alternativa en términos de SER de todos los algoritmos de la figura, llegando a cotas de $10^{-2}$. A priori la degradación existente en el resultado frente al resultado en la Figura 5.20 es normal, observando el resto de algoritmos y la tasa de variación del canal. Además no debemos olvidar que el algoritmo KNN dispone de una ventana deslizante con las últimas 200 muestras etiquetadas. Esto garantiza que si el algoritmo sigue la señal de variación, ofrecerá mejores prestaciones según vayan pasando las muestras de entrada, ya que olvidará el estado anterior erroneo del canal.

### 5.6.2.2 Gráfica SER frente a patrones de entrenamiento

Para mostrar los resultados de barrido en patrones, vamos a volver a utilizar las mismas señales de ejemplo que ya hemos comentado anteriormente.

Al igual que realizamos en la simulación de SER frente a SNR, vamos a realizar simulaciones de 5000 muestras de longitud, entre las que podemos dividir

dos etapas: hasta la muestra 1000 tenemos una etapa de entrenamiento estático inicial. De la muestra 1000 a 5000 tenemos entrenamiento dinámico, donde la variación de canal comienza en la muestra 2000.

En este caso, tenemos un punto por cada 500 muestras, y presentamos 3 iteraciones en el proceso de Montecarlo.

Cada punto de la gráfica corresponde con la evaluación de los algoritmos sobre una trama de test de la fuente de datos de longitud $10^5$ para garantizar la validez de los resultados mostrados.

A continuación realizaremos un recorrido por las señales de variación que hemos empleado mostrando los resultados que hemos obtenido. Las gráficas comienzan en el patrón 500 (ya que el intervalo de muestras es 500 y es el primer punto), que corresponde con la mitad del entrenamiento estático que se mostraba con un intervalo menor en la Figura 5.15. Se ha optado por incluir parte del entrenamiento estático en esta gráfica de entrenamiento dinámico con el objetivo de permitir una comparación mejor entre las dos partes de nuestro sistema. La verdadera variación de señal, comenzará a partir de la muestra 2000 de la gráfica.

- Canal senoidal suave: En la Figura 5.24 se puede ver el resultado de nuestra simulación, cuando la señal de variación de los coeficientes del sistema es una senoide suave, como la que se muestra en la Tabla 5.9.

  Con respecto al algoritmo de Volterra, vemos que tiene la peor respuesta de todas las alternativas bajo estudio, debido a los problemas comentados con anterioridad. A su favor, tenemos la estabilidad con la que trata la variación de canal.

  El algoritmo LMS mejora las prestaciones que ofrece en función del número de muestras, con lo que podemos ver una recta decreciente en SER. Según el algoritmo se va adaptando a la señal del canal, va reduciendo su tasa de error considerablemente. Probablemente ayude a este efecto el hecho de que la variación aplicada en el canal sea tan reducida en este caso.

Los algoritmos que mejores resultados nos proporcionan son RBF, KRLS y KNN, no teniendo diferencias significativas entre ellos.

En general los resultados que se muestran en la Figura 5.24 son en gran medida similares a los acontecidos en el entrenamiento estático, ya que la variación del canal no ha sido demasiado elevada.

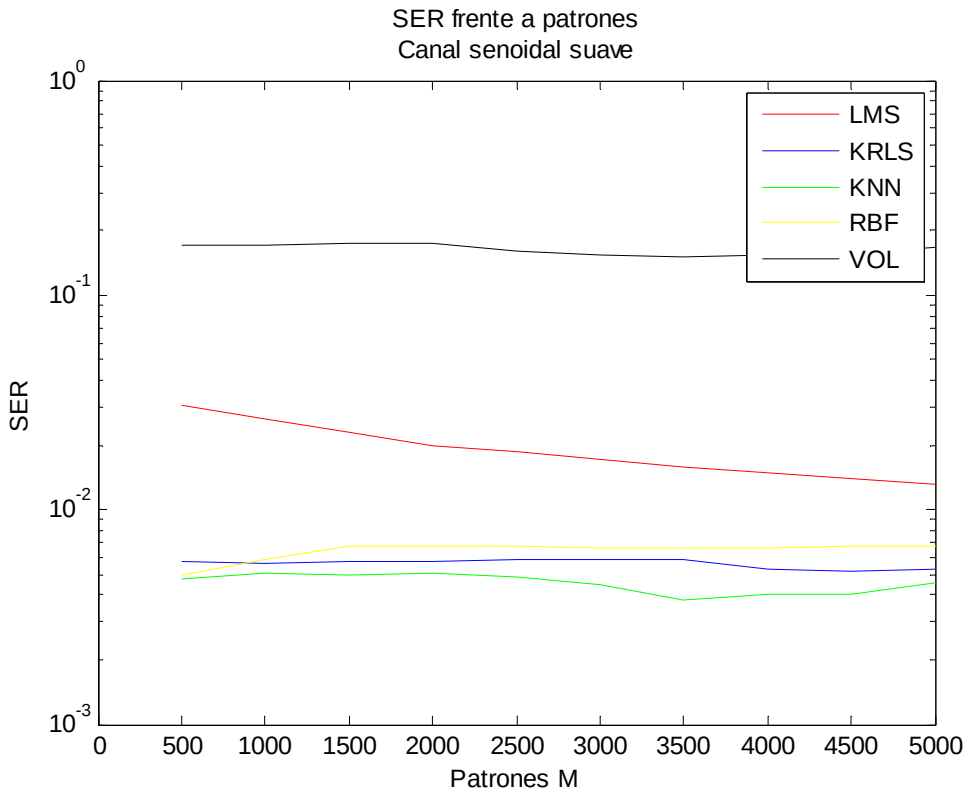



**Figura 5.24 - Gráfica SER frente a patrones para entrenamiento dinámico con señal senoidal suave**

- Canal senoidal medio: En la Figura 5.25 aparece el resultado de la simulación con la señal senoidal media.

Si bien el resultado de Volterra parece ser semejante al mostrado en la alternativa estática y en la senoidal suave, en este caso aparece una variación que nos lleva a una leve disminución de SER, nada más comenzar el cambio del canal. Esta alteración no es significativa.

El comportamiento decreciente que mostraba el algoritmo LMS en la Figura 5.24 se ve truncado en este caso, ante el cambio de señal, mediante el cual se mantiene una cota de SER, que sólo se reduce cuando el algoritmo se adapta a la variación.

Sin duda los casos más interesantes que aparecen en esta gráfica son RBF, KNN y KRLS, que son los que mejores prestaciones nos ofrecen.

Comenzando con el peor de ellos, tenemos el algoritmo RBF, que muestra un comportamiento cuasi estático, en los mismos níveles que la variación senoidal suave, pero con un aumento en la SER que aparece a partir de la muestra 2000 donde se comienza la variación de la señal. A pesar de eso, aparece un leve decremento entre las muestras 4000 y 5000 que llevan a pensar que el algoritmo se ha adaptado a la variación de señal, volviendo en este caso a la tasa de error que se obtiene en el entrenamiento estático.

El siguiente en prestaciones es KRLS, que al igual que le ocurría a RBF, presenta un remonte debido a la variación de canal, pero con un incremento menor al ofrecido por RBF. En este caso se puede corroborar de manera más precisa que el algoritmo llega a un estado estable en el tramo final del entrenamiento que asemeja las cotas de error al entrenamiento estático.

Por último presentamos el comportamiento del algoritmo KNN, que ofrece las mejores prestaciones de los algoritmos bajo estudio, al conseguir la mínima tasa de error. Durante la variación, aparece un decrecimiento bastante elevado de la tasa de error, que no puede ser mantenido, ya que entre las muestras finales del entrenamiento vuelve a ofrecer una cota de SER semejante a la estática. Es el algoritmo que mejores resultados presenta, pero en contra tiene la mayor tasa de variación durante el entrenamiento, lo que supone que sea el más sensible a la variación de los coeficientes de canal.

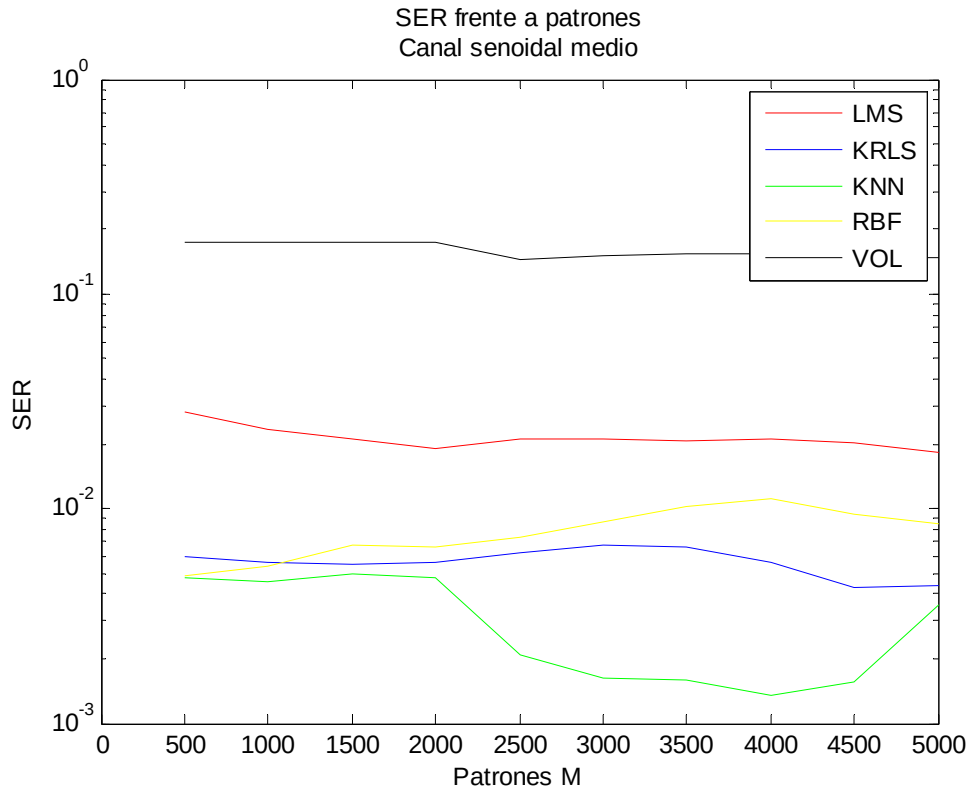



**Figura 5.25 - Gráfica SER frente a patrones para entrenamiento dinámico con señal senoidal medio**

- Canal escalón suave: Los resultados obtenidos se presentan en la Figura 5.26.

En ella se puede ver como el algoritmo de Volterra presenta las mismas prestaciones que para el caso senoidal medio, por lo que se mantiene invariante a la modificación de la forma de la señal.

Con respecto al algoritmo lineal LMS, vemos que el comportamiento empeora considerablemente, ya que a partir de la muestra 2000 se produce un incremento de la tasa de error de símbolo, lo que significa que el algoritmo no se recupera del escalón.

Por otro lado, tenemos el algoritmo RBF, que presenta el mismo comportamiento que en el caso senoidal medio, en el que encontramos una ligera subida de tasa de error a partir de la muestra 2000 donde realizamos la modificación de los coeficientes del canal. No obstante se observa como entre la muestra 3500 y 4000 el sistema se recupera, frenando el aumento de SER. De los tres mejores algoritmos (KRLS, RBF y KNN) es el que presenta peores prestaciones tras la variación.

El segundo mejor comportamiento nos lo ofrece el algoritmo KRLS, ya que aunque aparece una subida en su tasa de error a partir de la muestra 2000, se recupera rápidamente para ofrecer en la muestra 4000 la misma tasa de error que en el caso estático. Su comportamiento se puede considerar constante a pesar de la variación mostrada, ya que ésta es muy pequeña (menor incluso que la de RBF). EL comportamiento es semejante al mostrado en el caso de la señal senoidal media, con la única diferencia de que la variación en SER es algo mayor, aunque la recuperación ocurre en los mismos términos.

Por último tenemos el algoritmo KNN, que presenta el mejor comportamiento de los mostrados, al igual que ocurría en la variación senoidal media. Frente a la subida que experimenta el algoritmo KRLS, KNN ofrece una disminución de SER de los mismos márgenes, manteniendo su nivel durante toda la variación de la señal. Comparando con la variación senoidal media, tenemos que el decremento de SER es menor, al tener una variación de señal mayor. No obstante no aparece un efecto que si ocurría en el caso anterior, en el que el algoritmo incurría en un repunte entre las muestras 4000 y 5000 que igualaba la tasa de error presente en el entrenamiento estático, En este caso no aparece ese repunte, permaneciendo la reducción constante durante todo el proceso de variación de la señal.

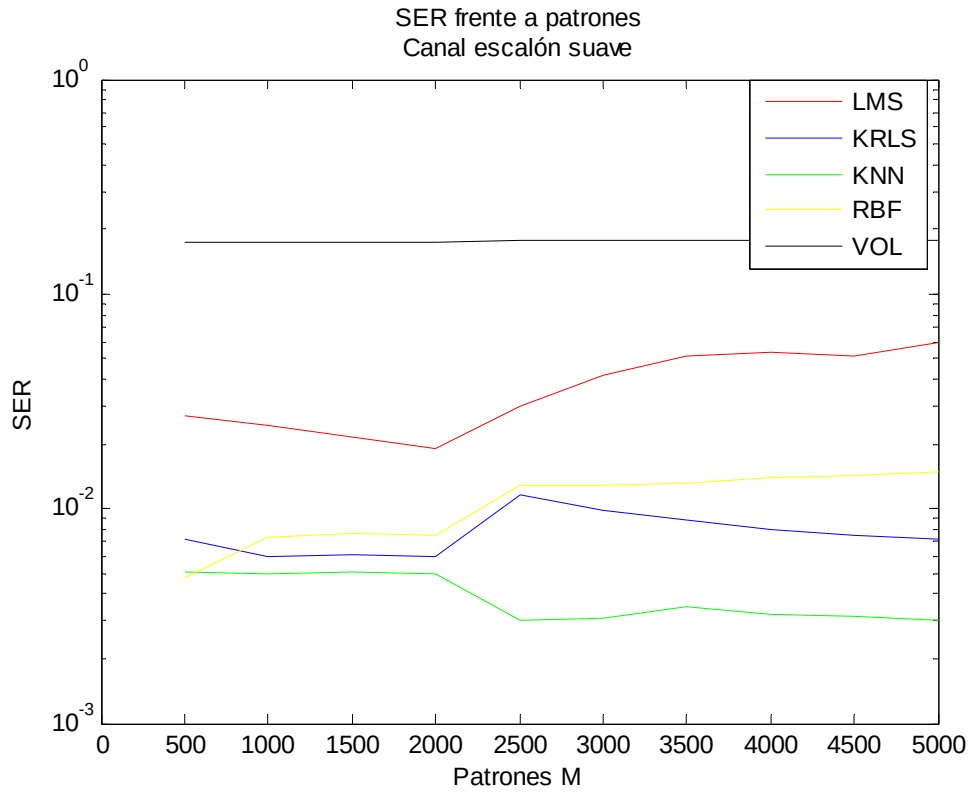



**Figura 5.26 - Gráfica SER frente a patrones para entrenamiento dinámico con señal escalón suave**

- Canal escalón medio: Para este último caso, obtenemos los resultados que aparecen en la Figura 5.27. En general vemos que los algoritmos tienen una variación muy elevada ya que para este caso el canal tiene un incremento muy elevado igualmente. En particular, las modificaciones de los resultados mostrados aparecen en un intervalo de 500 muestras (entre la 2000 donde se realiza la variación, y la 2500 donde se recuperan).

Para comenzar, tenemos el algoritmo de Volterra, el cual no es capaz de soportar la variación tan elevada de los coeficientes de canal, llegando a unas tasas de error de 1, lo cual nos dice que el algoritmo no converge.

Algo parecido le ocurre al algoritmo RBF, pero con una variación mayor, ya que pasa de ser el tercer mejor resultado de todos a no converger por la variación del canal aplicado.

En esta ocasión tenemos que la tercera mejor alternativa la representa el algoritmo lineal LMS, el cual sufre un empeoramiento

considerable en sus prestaciones, llegando a unas tasas de error elevadas, pero que mantiene constantes durante el entrenamiento dinámico.

A continuación tenemos el KRLS, que al igual que el resto presenta una variación considerable, que hace incrementar su SER. Al igual que realiza el LMS, ofrece una tasa constante una vez se consigue adaptar a la nueva situación, llegando incluso a tener una tasa suave decreciente durante el resto del entrenamiento. Un hecho importante, es que asemeja su comportamiento mucho más a la alternativa KNN que en el resultado de la Figura 5.26.

Por último, tenemos al KNN, que tiene una respuesta semejante al KRLS, con una tasa de error un poco más reducida durante todo el entrenamiento, pero que no es demasido significativa.

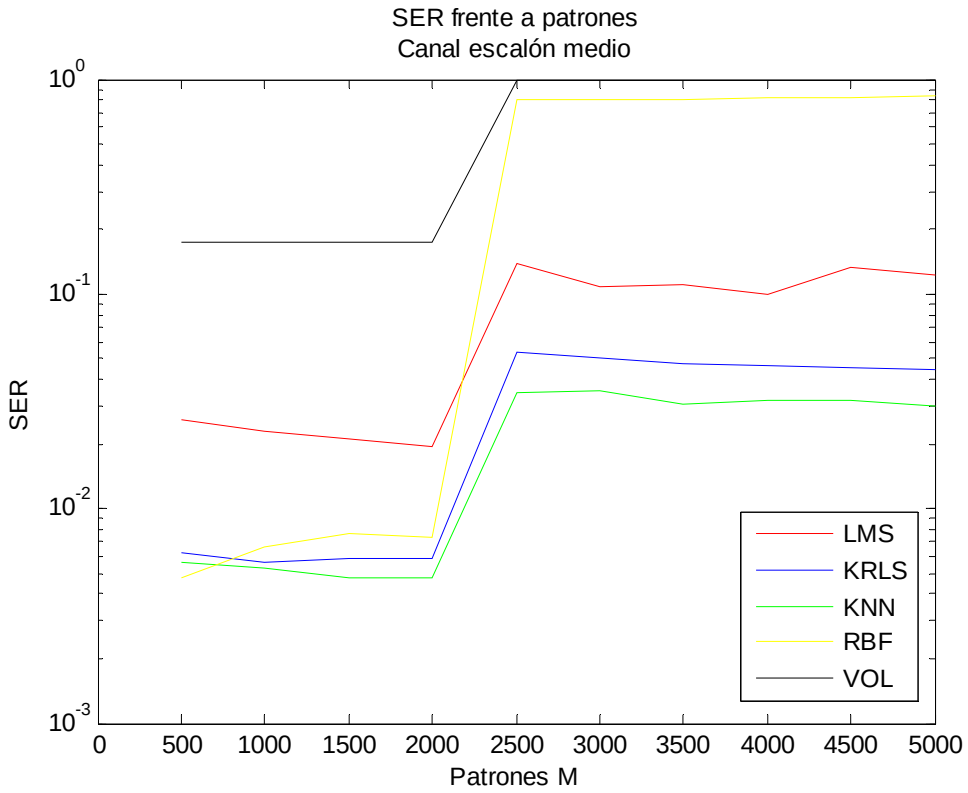



**Figura 5.27 - Gráfica SER frente a patrones para entrenamiento dinámico con señal escalón medio**

### 5.6.2.3 Relación de eficiencia para entornos dinámicos

A modo de realizar una comparativa entre el tamaño de la máquina del algoritmo KRLS implementado y el tiempo de procesamiento en comparación con el resto de algoritmos estudiados, hemos obtenido dos nuevos resultados: en el primero de ellos realizamos un barrido de tiempo de procesamiento de los algoritmos durante las muestras de entrenamiento. El segundo de ellos corresponde con el número de inserciones de muestras en el diccionario del algoritmo KRLS, o lo que es lo mismo, el tamaño de la máquina. Ambos resultados han sido extraidos para el peor caso posible de funcionamiento, correspondiente a un valor de SNR de 0 dB. En este punto, tenemos que el ruido alcanza las cotas más elevadas[26] , y nos proporciona el máximo tiempo de ejecución de todo el barrido de SNR, lo cual implica un mayor nivel de complejidad. Aunque a priori, parece ser una suposición justificada, comprobaremos la elección del púnto crítico (0 dB) de forma práctica, en el epígrafe de entrenamiento dinámico guiado por decisión.

Las dos gráficas extraidas, se han obtenido para todas las señales de variación introducidas en el presente epígrafe.

#### *5.6.2.3.1 Gráfica de tiempos de procesamiento*

Comenzamos revisando las gráficas de tiempos obtenidas para cada señal. Cada una de las tablas que mostraremos a continuación, se han dividido en dos elementos por motivos de claridad, ya que el resultado de KRLS es suficientemente elevado, como para provocar que el resultado del resto de algoritmos no sea visible. Por esa razón duplicamos el resultado obviando el resultado de KRLS, permitiendonos de esa manera realizar una comparativa entre los tiempos de proceso del resto de algoritmos.

Otro aspecto a tener en cuenta sobre los resultados que mostramos, son los picos de ejecución que muestran las gráficas, derivados principalmente del trabajo que ocasiona la impresión por pantalla de los resultados de la ejecución de la simulación. En el menor de los casos, también pueden estar justificadas por la

[26] El valor de 0 db corresponde al punto de mayor ruido de todo el barrido en SNR realizado en el proyecto, de 0 a 20 dB.

variación de la señal, aunque como veremos, este tipo de consecuencias aparecen principalmente en cambios lentos en la gráfica de tiempos. Por esta razón debemos detallar que la principal funcionalidad de estas gráficas es centrarnos en el valor medio, y no en los picos derivados de la propia ejecución en Matlab.

También debemos tener en cuenta la variabilidad de los resultados que se muestran, debida principalmente al equipo en el que se desarrolla la ejecución. Por este motivo se consideran útiles las gráficas a efectos de comparativa entre las prestaciones de los algoritmos, y no en valores absolutos.

- Canal senoidal suave: Las gráficas de tiempos obtenidas tras introducir una señal senoidal suave aparecen en la Tabla 5.11.

  Como podemos ver en la primera figura, el resultado de KRLS presenta un valor muy superior al resto de algoritmos bajo estudio, llegando a valores 5 veces mayores. No obstante este valor se mantiene constante durante todo el proceso de entrenamiento, a excepción de picos de procesamiento en los múltiplos de 1000 muestras, correspondientes a los momentos en que la salida por pantalla de la ejecución se mostraba. También, aunque en menor medida, se justifican por los cambiós de pendiente de los coeficientes de canal que se producen en esos puntos.

  Si observamos la segunda gráfica, podemos realizar una comparativa entre los resultados del resto de algoritmos. Observamos que aparecen igualemente, aunque en menor medida, los picos en los múltiplos de 1000 muestras. De los resultados mostrados, queda patente la menor complejidad del algoritmo LMS, como habiamos supuesto anteriormente al ser la alternativa lineal. Los siguientes algoritmos, que obtienen el doble de tiempo de ejecución que LMS, son RBF y Volterra, los cuales demuestran una similitud muy elevada. Por último, tenemos que el peor de los algoritmos, siempre por debajo de KRLS como hemos visto, corresponde con KNN que duplica el tiempo de RBF y Volterra.

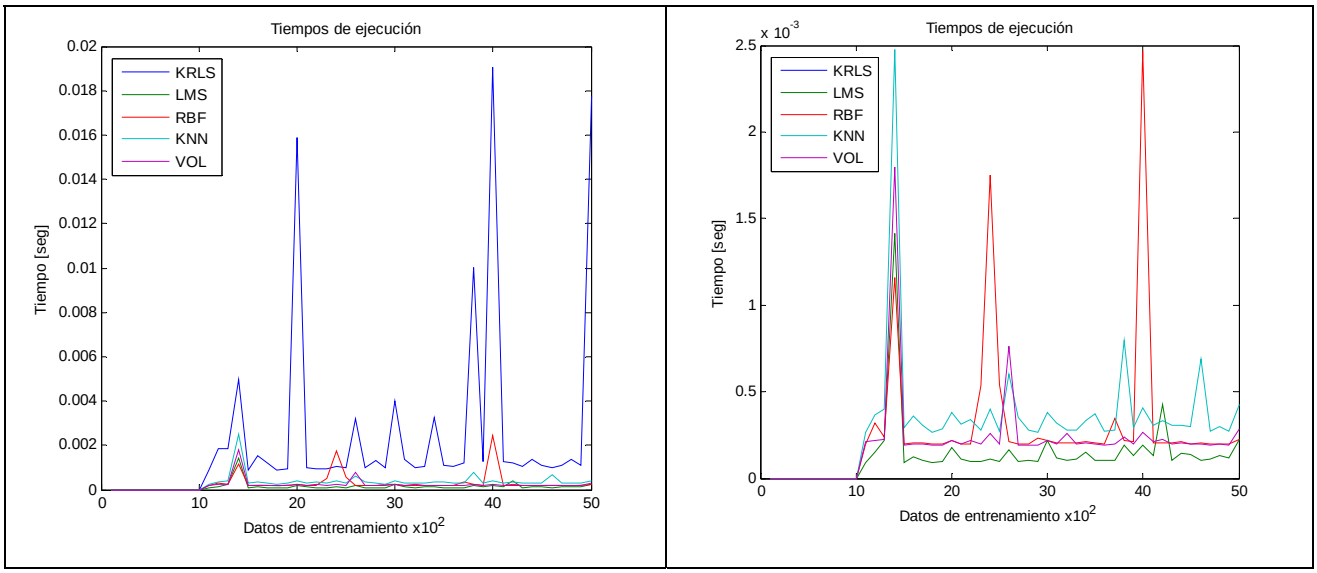



**Tabla 5.11 – Tiempos de ejecución para una señal de variación senoidal suave con SNR 0dB.**

- Canal senoidal medio: Los resultados obtenidos para esta señal de variación aparecen en la Tabla 5.12, con el mismo formato que en el caso anterior.

En este caso, vemos como KRLS sigue siendo la peor opción con claridad, con respecto al resto de algoritmos bajo estudio. La diferencia existente en este caso, es que sus prestaciones no se mantienen constantes durante todo el proceso de entrenamiento, como ocurría en el caso senoidal suave, ya que aparece una tendencia creciente de pendiente leve durante todo el proceso de entrenamiento.

Con respecto al resto de algoritmos, se repiten los mismos resultados de tiempos que en el caso de variación suave, con la excepción de una ligera pendiente de incremento en el caso de LMS y KNN, dejando a RBF y Volterra como las alternativas más estables de todas las mostradas.

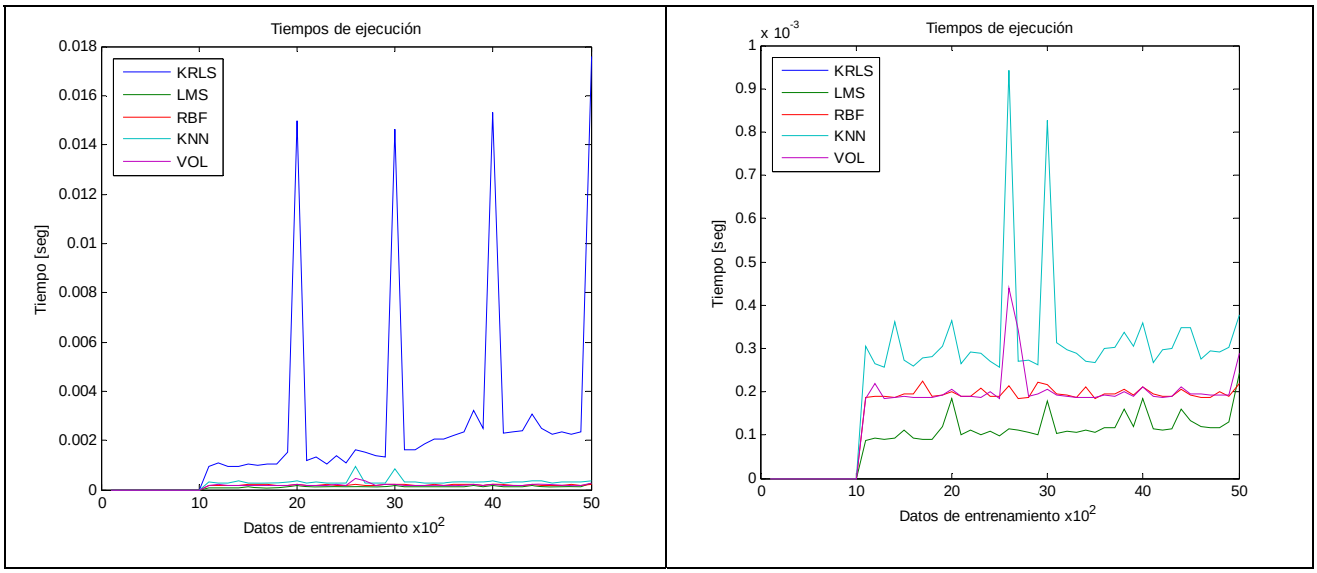



**Tabla 5.12 - Tiempos de ejecución para una señal de variación senoidal media con SNR 0dB.**

- Canal escalón suave: En la Tabla 5.13 vemos los resultados obtenidos para la señal de variación escalón suave.

En este caso, se observa una pendiente creciente para KRLS mucho más pronunciada que en las variantes senoidales, que comienza en el punto de inicio de la modificación de los coeficientes de canal (muestra 2000). Esta pendiente es mucho más importante que la que presentan el resto de algoritmos, lo cual deja patente las excesivas consecuencias que presenta el algoritmo KRLS ante la modificacón brusca del canal.

Con respecto al resto de algoritmos, volvemos a observar una mayor pendiente en el algoritmo LMS, que lleva a igualar sus tiempos de proceso a las variantes de RBF y Volterra, que en este caso se vuelven a mantener constantes con la variación de canal. El algoritmo KNN presenta igualemente un empeoramiento en sus prestaciones, aunque con una menor pendiente que la experimentada por LMS.

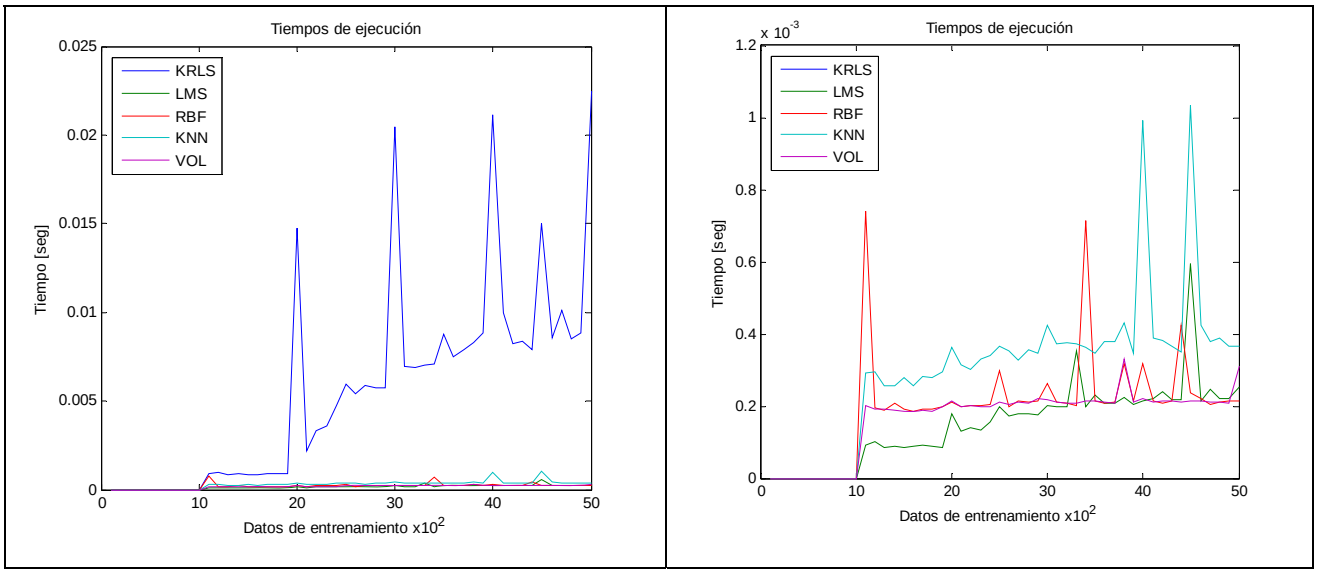



**Tabla 5.13 - Tiempos de ejecución para una señal de variación escalón suave con SNR 0dB.**

- Canal escalón medio: Los resultados obtenidos aparecen en la Tabla 5.14.

En ella podemos ver como la pendiente de KRLS sigue incrementandose, llegando en este caso a valores muy elevados de tiempo. Esto no hace más que corroborar a efectos prácticos la problemática del algoritmo tal y como se comentó en el epígrafe 5.6.2.1. Este aspecto hace que las simulaciones del algoritmo sean muy costosas a efectos de tiempo de procesamiento, y en consecuencia en complejidad.

Con respecto al resto de algoritmos, vemos como el algoritmo LMS tiene una pendiente mayor que la mostrada en la alternativa escalón suave, que hace que su tiempo de procesamiento alcance de una manera más rápida al obtenido por RBF y Volterra, que siguen manteniendose constantes. Por otro lado, tenemos que el algoritmo KNN sigue presentando los mismos tiempos de ejecución que los presentes en la variante escalón suave (una pequeña pendiente creciente en la curva), lo cual nos permite decir que sus prestaciones son prácticamente constantes.

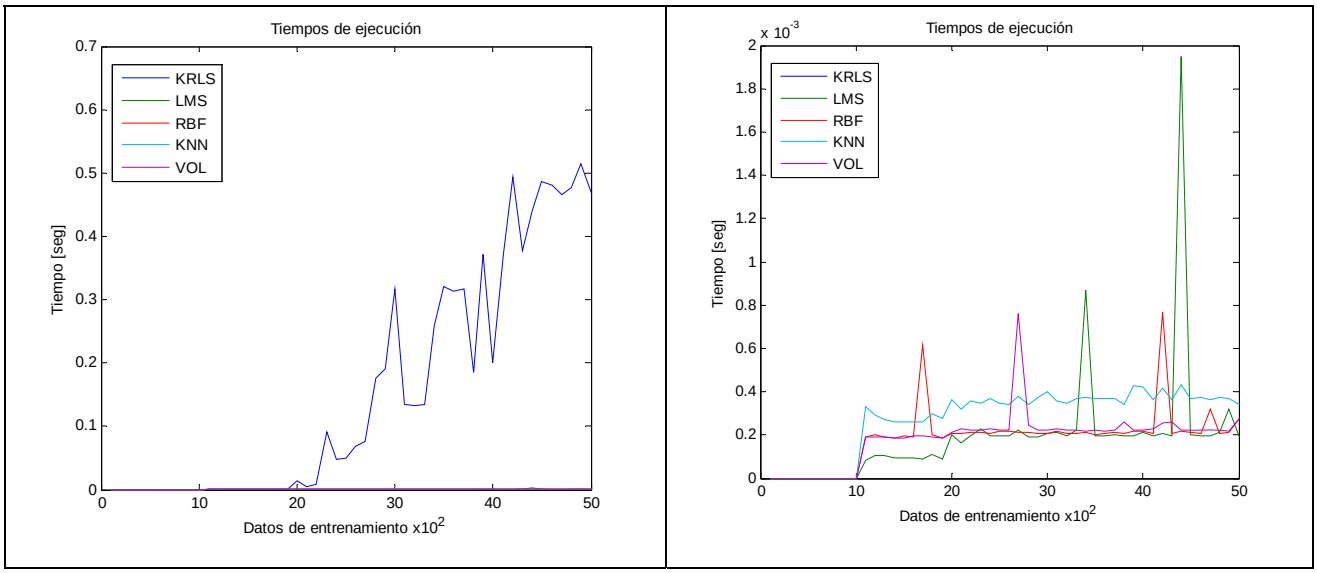



**Tabla 5.14 - Tiempos de ejecución para una señal de variación escalón medio con SNR 0dB.**

### *5.6.2.3.2 Gráfica de tamaño de la máquina del algoritmo KRLS*

El siguiente resultado que hemos obtenido es referente al tamaño del diccionario del algoritmo KRLS, en referencia al número de muestras que se encuentran incluidas en el mismo, y que nos sirven para establecer la comparación ALD, necesaria para definir las nuevas muestras en función de las ya almacenadas.

En esa dirección hemos obtenido la variación del tamaño de ese diccionario durante el proceso de entrenamiento dinámico para una SNR de 0dB. considerada como el peor caso posible, de entre los valores del barrido de SNR realizado en el presente proyecto fin de carrera.

El motivo por el que mostramos únicamente el tamaño del algoritmo KRLS, es porque es el único que consideramos de núcleo variable, ya que el resto no crecen con respecto al tiempo y como tal pueden ser considerados como constantes (este aspecto se puede comprobar revisando la especificación de cada algoritmo presente en el Capítulo 4).

Estas gráficas ya han sido utilizadas (Figura 5.21 y Figura 5.23) para justificar la problemática encontrada con las señales de variación escalón medio en el entrenamiento de KRLS, y que hacía presentar unos tiempos de

procesamiento muy elevados, tal y como hemos podido ver en el apartado anterior. Volvemos a utilizarlas en este caso, para realizar una comparativa entre el tamaño de la máquina ante el resto de señales de variación.

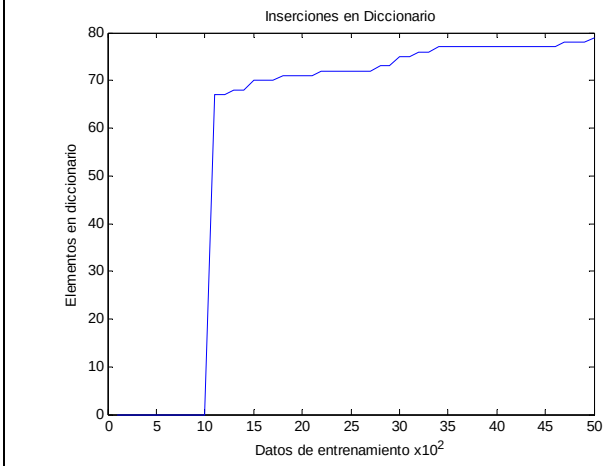
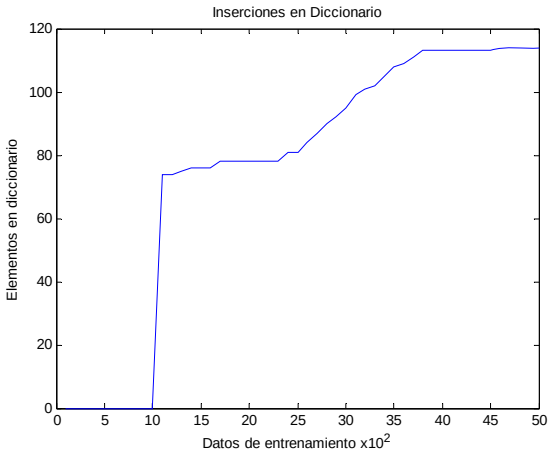
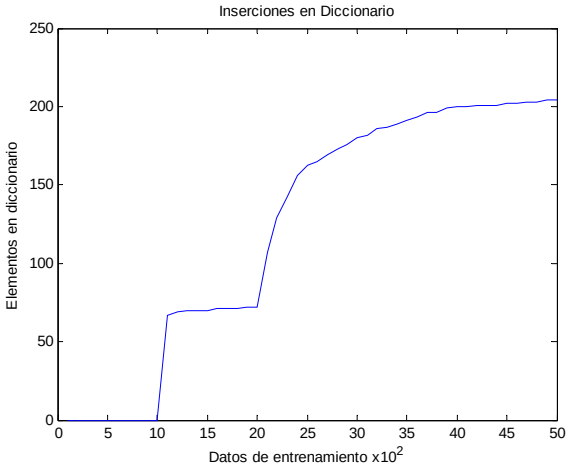
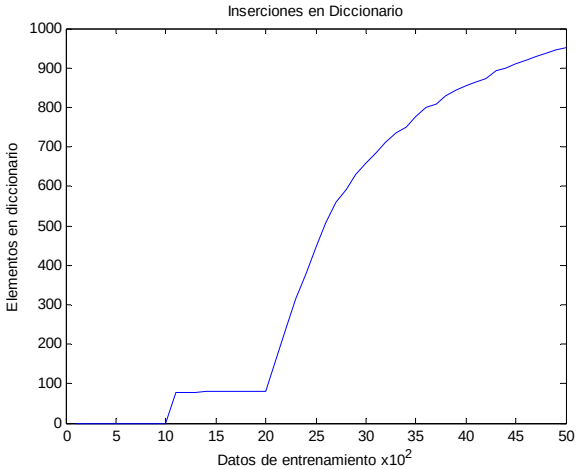
| | | Variación | |
|---|---|---|---|
| | | Suave | Media |
| Tipo | Senoidal |  |  |
| | Escalón |  |  |

**Tabla 5.15 – Inserciones en el diccionario del algoritmo KRLS durante el proceso de entrenamiento para SNR = 0dB.**

El resultado presente en la Tabla 5.15 nos muestra de manera clara la diferencia existente en el comportamiento del algoritmo KRLS ante las diferentes señales de variación.

Podemos ver que para la señal senoidal, tenemos que el algoritmo se adapta rápidamente, con unas pendientes lineales que hacen llegar a un estado estable a la máquina. En contra, tenemos los resultados de la señal escalón, que presentan una pendiente exponencial, llegando al punto de no encontrar un estado estable, como es el caso de la variante media. Este aspecto, junto con el elevado

número de miembros del diccionario hace disparar los tiempos de procesamiento del algoritmo KRLS en comparación con el resto de alternativas como vimos en el resultado anterior.

### 5.6.3 Esquema de igualación de canal basado en entrenamiento guiado por decisión

Para finalizar los resultados de nuestro proyecto fin de carrera, y tras haber comentado el entrenamiento dinámico, vamos a modificar el modelo de entrenamiento de nuestros experimentos, introduciendo el esquema de aprendizaje guiado por decisión que detallamos en el punto 5.2.4.

En este caso mantenemos el entrenamiento estático inicial del sistema de longitud M = 1000 muestras, para garantizar la convergencia de los algoritmos hacia un estado estable, pero modificamos la parte dinámica comentada en el punto anterior. Dicha modificación radica principalmente en un cambio en el proceso de entrenamiento de los algoritmos, ya que en el caso anterior utilizabamos la entrada conocida del sistema $u_t$, mientras que en esta ocasión empleamos para el entrenamiento de nuestros algoritmos una estimación previa de la entrada realizada con los pesos de la iteración anterior. Para decirlo de otro modo, en la iteración 't' de nuestro sistema, estimamos la entrada del sistema $\hat{u}_t$ con los pesos del sistema $\omega_{t-1}$ y la salida del canal en el instante 't', $y_t$. Con esta estimación entrenamos nuestro sistema para obtener los nuevos pesos, $\omega_t$, que serán empleados para la estimación de la iteración siguiente y para el cálculo de la tasa de error. En el calculo de dicha tasa de error en el instante 't', utilizamos la entrada del sistema real, $u_t$, después de haber entrenado el sistema con la estimación $\hat{u}_t$. Todo este proceso puede verse en la expresión

5.7, donde se muestran las relaciones comentadas, siendo 'f','g' y 'k' funciones dependientes del algoritmo en cuestión sobre el que realicemos el entrenamiento.

$$t \rightarrow \begin{cases} \hat{u}_t = f\left(\omega_{t-1}, y_t\right) \\ \omega_t = g\left(\hat{u}_t, y_t\right) \\ SER_t = k\left(u_t, y_t\right) \end{cases} \qquad \textbf{5.7}$$

Con respecto a la modificación del canal, vamos a emplear las mismas señales comentadas en el apartado anterior en la Tabla 5.9, con el fin de poder realizar comparaciones entre los resultados obtenidos con el entrenamiento estático, dinámico, y el entrenamiento dinámico guiado por decisión.

Para todos los resultados del epigrafe hemos intentado repetir los mismos experimentos que en el caso dinámico, para poder considerar las comparaciones realizadas como conclusiones válidas.

En esa dirección, hemos realizado entrenamientos de 5000 muestras considerando como entrenamiento estático hasta la muestra 1000, siendo el entrenamiento dinámico desde la 2000 hasta la 5000. El intervalo que queda por comentar, comprendido entre la muestra 1000 y 2000 se considera parte del entrenamiento dinámico, pero con los mismos valores de coeficientes que para el caso estático. Esto permite definir de manera más precisa el estado estático de la respuesta del algoritmo, previo a la modificación, así como el comportamiento del nuevo esquema ante un canal estático.

A modo de especificar de manera más precisa la modificación de canal, debemos tener en cuenta que tenemos un periodo de entrenamiento puramente estático, ya que el intervalo existente entre la muestra 1000 y 2000 que tiene los coeficientes de canal sin modificación ya se encuentra con un entrenamiento guiado por decisión, es decir, ese periodo podríamos denominarlo como entrenamiento estático guiado por decisión. Su objetivo es el de garantizar la convergencia previa del algoritmo. Esto supone una diferencia considerable con el entrenamiento anterior, que se verá plasmada en los resultados que mostraremos.

Tal y como se hizo en el apartado anterior, realizamos un suavizado de las gráficas de salida con un promedio de Montecarlo de 3 iteraciones. Para caracterizar completamente el sistema, tenemos que los parámetros que rigen el comportamiento de los algoritmos son semejantes a los mostrados en la Tabla 5.5 para un entrenamiento estático, con el objetivo de que los resultados sean comparables. Para asegurar la veracidad de los resultados mostrados, se han realizado con secuencias independientes de test de $10^6$ muestras (garantizamos que es una década mayor que la inversa de la menor tasa de error que mostramos).

Al igual que en el caso anterior, vamos a examinar las gráficas de SER frente a SNR, y de SER frente a patrones, pero en esta ocasión vamos a dar un repaso por las gráficas de error en el entrenamiento dinámico guiado por decisión, como ya hicimos en el caso estático. Introducimos este resultado, porque la convergencia de los algoritmos es un punto muy importante, ya que de no conseguirla, se podrían presentar problemas de propagación de errores, al estimar la entrada con un estado erroneo.

Este epigrafe se considera el resultado principal del presente proyecto fin de carrera.

#### 5.6.3.1 Gráfica SER frente a SNR

La configuración elegida para estas gráficas es semejante a la mostrada para el caso dinámico, tanto en longitud, parámetros, etc, por lo que pasamos a detallar los resultados obtenidos con nuestras señales de variación (Tabla 5.9).

- Canal senoidal suave: Los resultados que obtuvimos con esta señal de variación se muestran en la Figura 5.28. Debemos remarcar la diferencia de comportamiento de las gráficas con respecto a la alternativa de entrenamiento dinámico, principalmente motivados por la diferencia de esquema y del aprendizaje del mismo.

  Es evidente para el resultado del algoritmo de Volterra que no se adapta convenientemente al entrenamiento guiado por decisión, ya que modifica el punto de partida de la gráfica al valor 1, lo que implica que no no converge inicialmente para valores de SNR de 0 a 4 dB, ofreciendo resultados sin cotas (obtenemos valores que tienden a infinito). A partir de ahí se obtienen resultados equiprobables en las secuencias de test que se introducen. Esto viene a significar que el algoritmo ofrece resultados acotados, pero que no converge hacia una solución correcta, ya que se obtienen valores de SER de 0.5.

  Con respecto al algoritmo lineal LMS, tenemos un comportamiento que mejora al mostrado en la alternativa de entrenamiento dinámico. A

efectos cuantitativos, tenemos que la minima tasa de error se mejora en una década, lo cual es un valor considerable, para valores de SNR mayores a 12 dB. Para valores de SNR menores de 12dB, tenemos que LMS iguala el resultado que ofrecen las alternativas no lineales. Esto supone una mejora sobre la alternativa dinámica, ya que el incremento de prestaciones del algoritmo hace subir el punto de ruptura con los otros algoritmos de los 10dB, correspondientes al entrenamiento dinámico ya comentado, a los 12 dB actuales.

Los algoritmos RBF y KRLS ofrecen las mismas prestaciones, siendo similares aproximadamente a las mostradas en el entrenamiento dinámico.

Con respecto al algoritmo KNN, tenemos las mismas prestaciones que KRLS y RBF a partir de 10 dB, no obstante, vemos que para relaciones señal a ruido menores, presenta variaciones, lo cual nos hace pensar que en este tipo de entornos es muy dependiente de la SNR de las muestras de entrada al sistema.

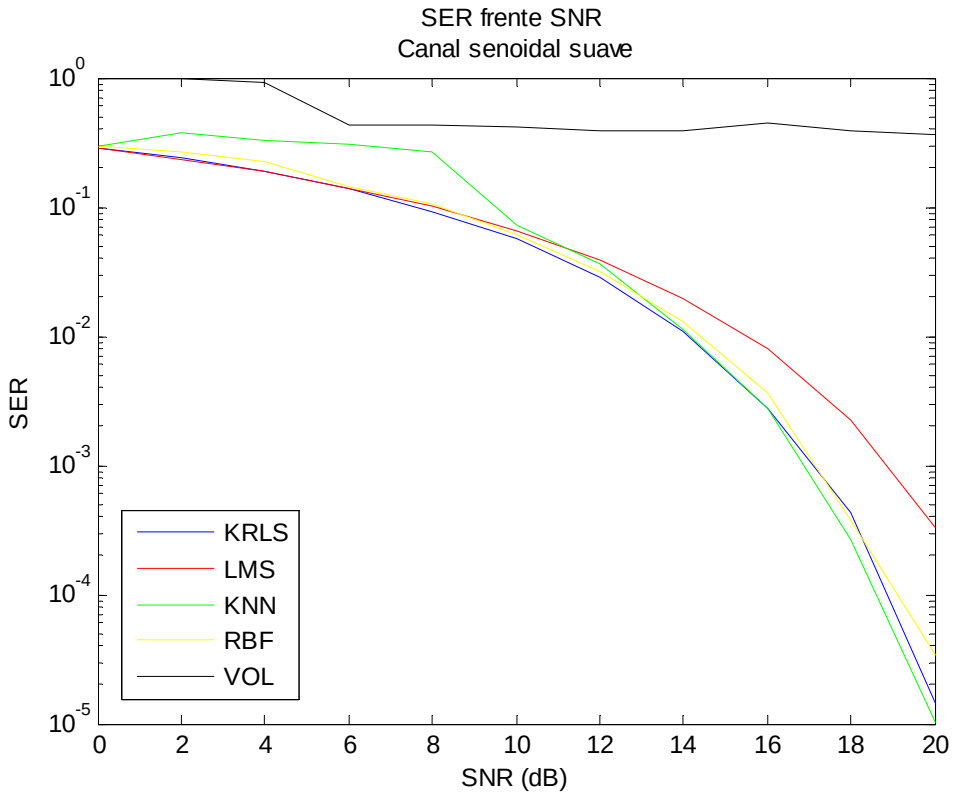



**Figura 5.28 – SER frente a SNR para canal senoidal suave y entrenamiento dinámico guiado por decisión**

- Canal senoidal medio: Los resultados que obtuvimos en la simulación se muestran en la Figura 5.29.

Comenzando con el algoritmo de Volterra vemos que al igual que ocurría con la señal senoidal suave, tenemos que no converge sin cota hasta los 4dB, y a partir de ahí no converge a una solución correcta. Como era de esperar no ha mejorado su respuesta ante una señal de variación mayor.

El algoritmo LMS empeora un poco con respecto a la variación senoidal suave, aunque no obstante, debido al mayor empeoramiento del resto de algoritmos, asemeja aún más su respuesta a la que proporcionan las alternativas no lineales, llegando al punto de obtener aproximadamente los mismos resultados con una significativa menor complejidad.

Al igual que ocurría en el caso anterior, tenemos que KRLS y RBF vuelven a tener las mismas curvas aproximadamente, aunque en esta ocasión empeoran sus resultados algo más de una década, lo cual nos lleva a pensar que tienen una fuerte dependencia a la variación de los coeficientes de canal (al menos mayor que la que presenta la alternativa no lineal).

El mayor empeoramiento lo representa el algoritmo KNN, ya que la degradación que se presentaba hasta los 10dB en el caso anterior, se mantiene ahora durante toda la curva de SER frente a SNR, empeorando incluso para SNR mayores de 12 dB. Viendo este resultado, destacamos, que el algoritmo KNN parece no soportar demasiado bien la variación continuada de los coeficientes del canal.

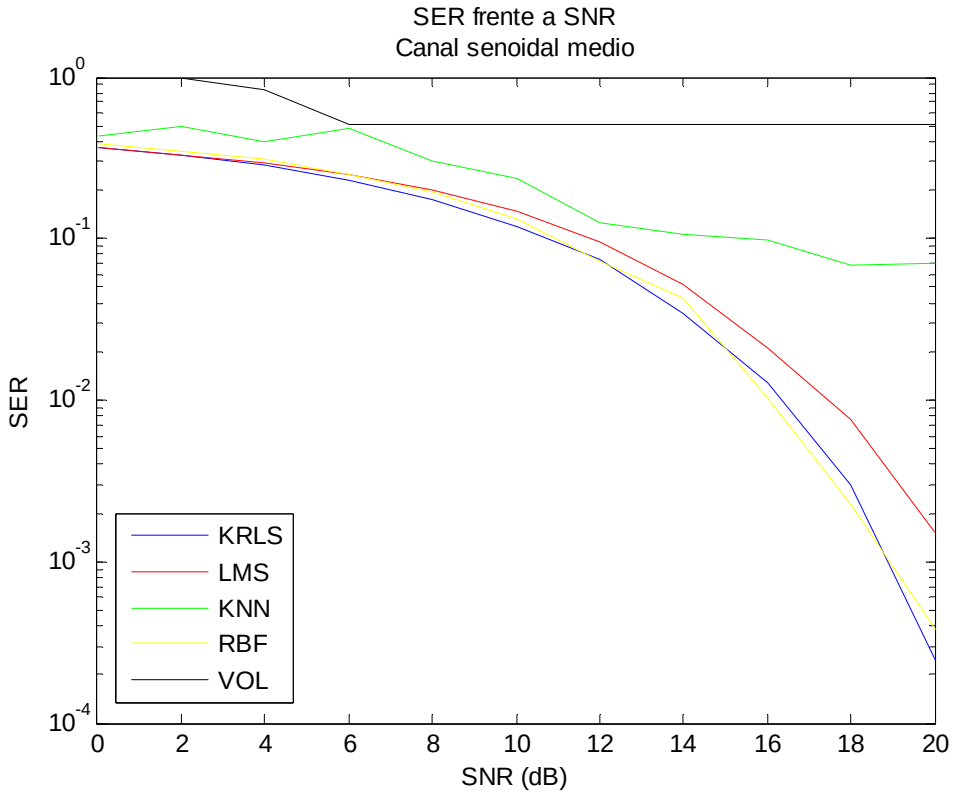



**Figura 5.29 – SER frente a SNR para canal senoidal medio y entrenamiento dinámico guiado por decisión**

- Canal escalón suave: En la Figura 5.30 vemos los resultados obtenidos para un proceso de entrenamiento guiado por decisión con una señal variante escalón suave. A primera vista se puede determinar que el escenario que se presenta es prácticamente complementario al mostrado con la variación senoidal. Un apunte a tener en cuenta, que se repite con el entrenamiento dinámico de escalón suave, son los problemas de convergencia de los algoritmos, y en particular del KRLS, al aumentar de manera considerable los elementos presentes en el diccionario. En esta ocasión, las cotas de tiempo de ejecución pueden ser asumibles, por lo que se ha mantenido la misma configuración que en el resto de simulaciones al igual que se realizó en el entrenamiento anterior para el esquema de aprendizaje dinámico.

Lo primero que podemos ver en la gráfica mostrada, es que el trazo de Volterra no aparece. Esto es debido a que tal y como ocurría en la variación senoidal, tenemos que ofrece un resultado no acotado, lo cual implica que el algoritmo no converge durante todo el barrido de SNR. En

este punto nuestro algoritmo es inservible, para nuestros propositos de variación escalón.

El algoritmo LMS presenta un comportamiento muy diferente al caso senoidal, ya que empeora significativamente el resultado anterior, perdiendo incluso 2 décadas de SER. En este punto, tenemos que la alternativa lineal no es viable para variaciones bruscas de los coeficientes de canal, ya que no recupera el estado de convergencia (lo veremos de una forma más clara posteriormente con las gráficas de error). No obstante, sigue igualando los resultados de los dos mejores algoritmos no lineales hasta los 8dB.

El segundo mejor algoritmo es RBF, que al igual que LMS presenta una degradación de 2 décadas, principalmente desde la cota de 8dB, donde los algoritmos separan sus resultados.

KRLS presenta un resultado no convergente y totalmente degradado, ya que no es capaz de recuperarse del estado de variación, para alcanzar un estado estable. Revisaremos este punto más adelante con las gráficas de error. Su comportamiento es claramente peor que la alternativa lineal.

La mejor opción la ofrece el algoritmo KNN (al igual que ocurría en el entrenamiento dinámico), al contrario que en la variación senoidal. En este sentido, no solo se reducen las variaciones que aparecían hasta los 4dB, sino que se obtienen los resultados del canal senoidal suave, a pesar de que esta variación supone una tasa mayor de una década porcentual sobre la anterior.

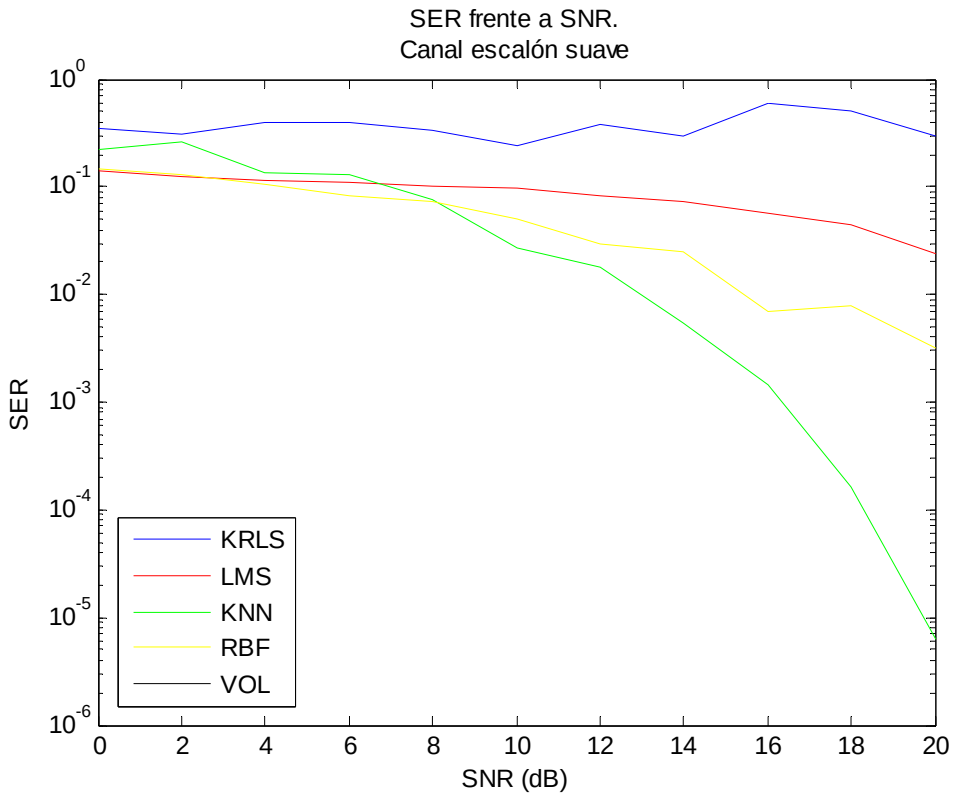



**Figura 5.30 – SER frente a SNR para canal escalón suave y entrenamiento dinámico guiado por decisión**

- Canal escalón medio: En la Figura 5.31 aparecen los resultados para una señal de variación escalón medio. Al igual que ocurría en el caso dinámico, se han tenido problemas en la convergencia del algoritmo KRLS, ya que el número de elementos del diccionario se disparaba a cotas que hacían impracticables los tiempos de ejecución de las simulaciones. Por esa razón se ha optado por repetir la configuración propuesta en el entrenamiento dinámico, centrando el número de muestras totales del entrenamiento a 2250 muestras, siendo esta cota un compromiso entre tiempos de ejecución y prestaciones de los algoritmos.

En este caso volvemos a comprobar, tal y como era de esperar, que el algoritmo de Volterra sigue sin obtener resultados acotados ya que no aparece su trazo en la gráfica.

En esta ocasión vemos como la respuesta del algoritmo LMS se mantiene semejante al caso escalón suave a pesar de aumentar de manera considerable la tasa de variación. Esto nos da una idea de la

consistencia del algoritmo frente a las alternativas no lineales que mostramos.

Por otra parte, tenemos que el algoritmo KRLS, mantiene exactamente los mismos resultados mostrados en la variación escalón suave, pero con menor variación. Este hecho no merece mayor consideración, mas allá de que el algoritmo no diverge a valores infinitos, ya que la cota de error es aproximadamente 0.5 durante todo el barrido de SNR.

Algo a tener en cuenta es la mayor degradación que sufre RBF, empeorando los resultados de KRLS (en el caso anterior, teniamos que mejoraba a KRLS considerablemente), con cotas mayores a 0.5 de SER, lo cual nos lleva a pensar que tiene problemas de convergencia.

Al igual que ocurría en la variación suave, tenemos que la mejor alternativa, la presenta el algoritmo KNN, ya que parece adaptarse de mejor forma que el resto a cambios bruscos. En este caso empeora significativamente sus prestaciones (en torno a 3 décadas) aunque es justificable en terminos del incremento de variación aplicado. Para este caso en particular no presenta mejores prestaciones significativas frente a la alternativa lineal.

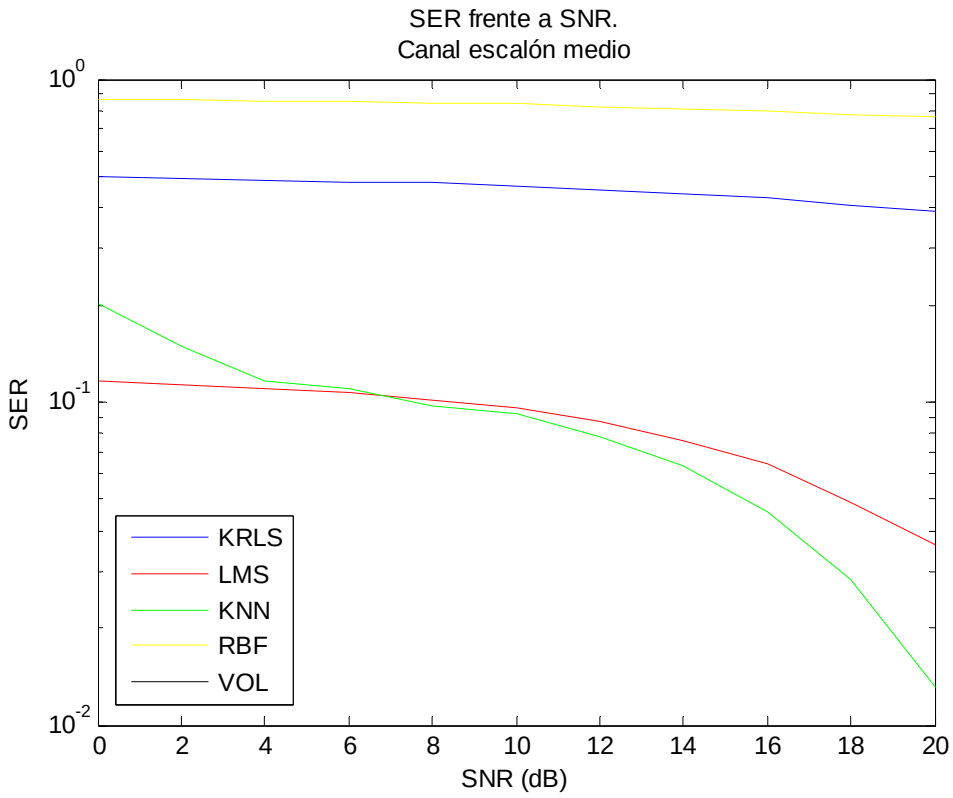



**Figura 5.31 – SER frente a SNR para canal escalón medio y entrenamiento dinámico guiado por decisión**

### 5.6.3.2 Gráfica SER frente a patrones de entrenamiento

En este punto vamos a continuar con el trabajo expuesto en el punto anterior, volviendo a utilizar de nuevo las mismas señales de variación.

Con respecto a la configuración de las simulaciones correspondientes a los resultados que mostramos, debemos tener en cuenta que es exactamente igual que la comentada en el caso de entrenamiento dinámico, y semejante igualmente a los resultados de SER frente a SNR mostrados anteriormente para el entrenamiento que nos ocupa.

A modo de recordatorio, debemos mencionar que las gráficas tienen una longitud de entrenamiento de 5000 muestras, siendo las 1000 primeras correspondientes a entrenamiento púramente estático, y las siguientes al modelo de entrenamiento guiado por decisión. La variación de los coeficientes comienza en la muestra 2000 del entrenamiento para garantizar un estado de convergencia de los algoritmos.

Examinamos las diferentes señales de variación bajo estudio:

- Canal senoidal suave: En la Figura 5.32 vemos el resultado obtenido ante la señal de variación senoidal suave en nuestro sistema.

De un primer vistazo, vemos el mal comportamiento que presenta el algoritmo de Volterra ante el paso de entrenamiento estático a guiado por decisión a pesar de que en ese periodo no se produce ninguna variación en los coeficientes del canal. Esto aparece entre la muestra 1000 y 1500, donde vemos un incremento en la tasa de error de media década, el cual lleva al algoritmo a valores inservibles para cualquier aplicación practica, ya que como se puede comprobar, el algoritmo no converge a una solución correcta al ofrecer una SER de 0.5. La tasa de error se mantiene en este valor durante todo el periodo de variación de la señal.

Por otro lado, tenemos el algoritmo LMS, que presenta un comportamiento adecuado al cambio de entrenamiento, al igual que al cambio de coeficientes con la señal senoidal. Ofrece un resultado no demasiado bueno, pero según aumenta el entrenamiento produce una clara tendencia a reducir la tasa de error.

Con respecto a los algoritmos KRLS, KNN y RBF, vemos que tienen las mismas prestaciones, siendo KRLS el que ofrece mayor estabilidad durante todo el proceso de entrenamiento, tanto estático como dinámico (RBF ofrece un pequeño incremento al iniciar el entrenamiento guiado por decisión, al igual que KNN, aunque al final consiguen converger a soluciones correctas.)

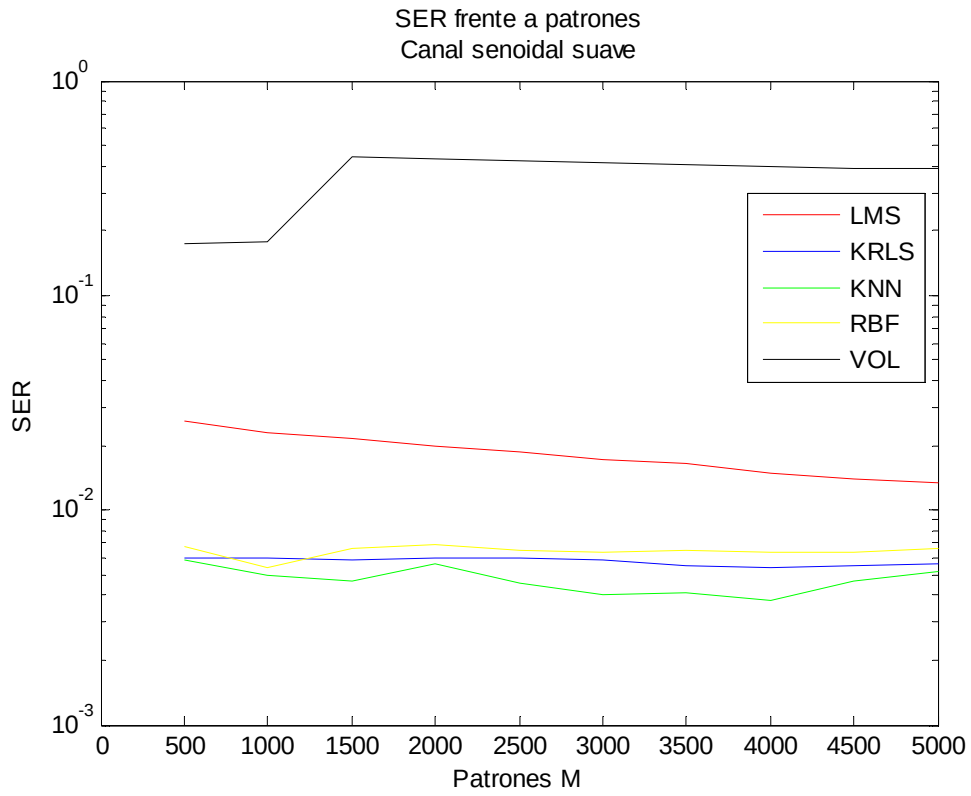



**Figura 5.32 – SER frente a patrones para canal senoidal suave y entrenamiento dinámico guiado por decisión**

- Canal senoidal medio: La salida obtenida para la entrada senoidal media aparece en la Figura 5.33.

A primera vista tenemos la modificación del algoritmo de Volterra con respecto a la señal senoidal suave, ya que aunque sigue sin converger (era de esperar que no convergiera en este caso tampoco, ya que tenemos una tasa de variación mayor), tenemos que presenta una mayor variabilidad. El empeoramiento que se presentaba ante el cambio de entrenamiento sigue presente en este caso, pero al introducir la variación parece que el algoritmo intenta converger, pero acaba el entrenamiento, en los mismos valores de SER de 0.5.

El comportamiento del LMS es semejante al obtenido al aplicar la señal senoidal suave, si bien parece no tener una pendiente con tendencia tan decreciente.

Con respecto a los otros algoritmos no lineales, tenemos que la menor tasa de error la presenta el algoritmo KNN durante la mayor parte del entrenamiento dinámico. No obstante eso no es del todo deseable, ya que al finalizar el entrenamiento aparece un incremento en la SER que produce que la tasa de error se iguale con la del KRLS que se había mantenido aproximadamente constante. Esto parece apuntar a que el algoritmo KNN es muy dependiente de las variaciones de los coeficientes, y que no llega a adaptarse completamente a ellas.

El algoritmo KRLS presenta la mejor opción bajo nuestro punto de vista, ya que no produce ninguna modificación entre los diferentes modelos de entrenamiento, presentando únicamente una variación muy pequeña (del orden del algoritmo LMS) en sus prestaciones, y alcanzando en cualquier caso, un estado estable al finalizar las muestras del entrenamiento dinámico.

Por último, tenemos el resultado obtenido para RBF, que es algo superior al de KRLS, sufriendo un incremento en SER en el cambio de entrenamiento, y en el comienzo del mismo. Podemos decir en su favor, que al finalizar la etapa dinámica, parece que tiende a alcanzar la tasa de error de KRLS y KNN.

A pesar de todo lo dicho, los tres algoritmos no lineales comentados: KRLS, RBF y KNN, producen resultados semejantes al finalizar el entrenamiento.

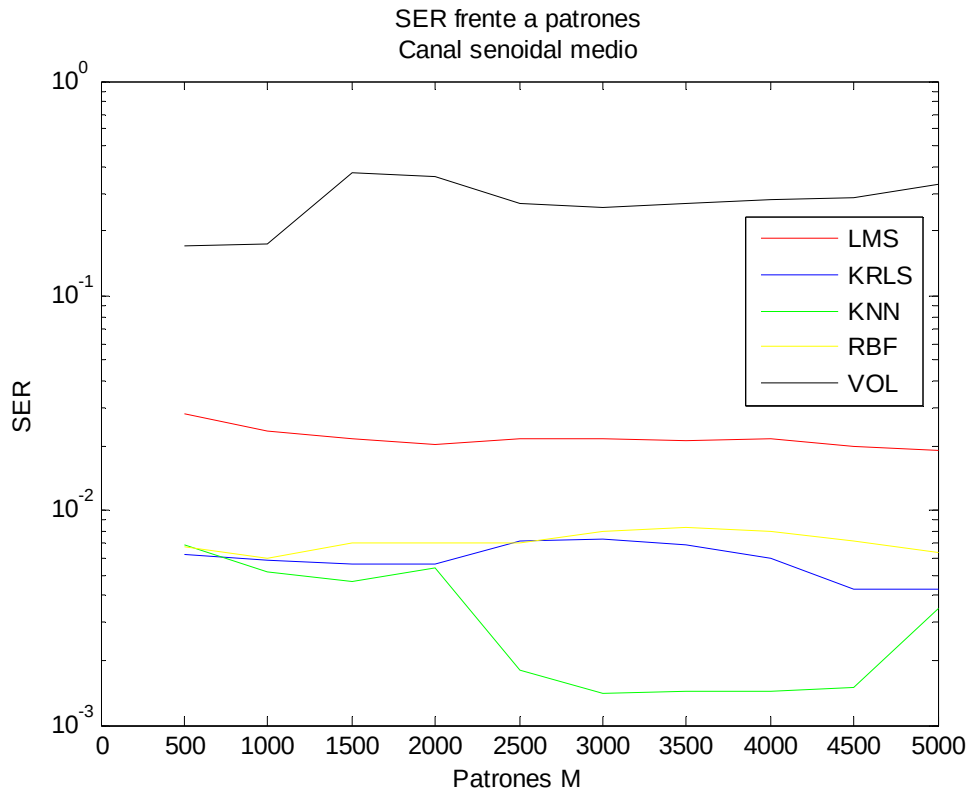



**Figura 5.33 - SER frente a patrones para canal senoidal medio y entrenamiento dinámico guiado por decisión**

- Canal escalón suave: En la Figura 5.34 vemos el resultado del sistema de entrenamiento dinámico guiado por decisión ante una entrada escalón suave.

En ella podemos ver como el algoritmo de Volterra presenta serias dificultades ante el cambio a entrenamiento estático guiado por decisión, que lo lleva a no conseguir una solución adecuada. Esta respuesta empeora cuando introducimos la señal de variación, ya que lo que conseguimos es que el algoritmo diverga.

Algo muy alarmante en la gráfica es el gran empeoramiento que presenta el algoritmo KRLS ante la introducción de la señal de variación (aunque se mantiene estable en el cambio de entrenamiento). El resultado que obtenemos se puede considerar como que el algoritmo no converge a una solución válida ante esta señal de variación.

El algoritmo LMS presenta una tendencia de aumento de SER ante la variación de los coeficientes. Al igual que en KRLS, tenemos que en el

cambio de entrenamiento, no se produce ninguna variación en el estado de salida de este algoritmo. El resultado que obtenemos, si bien converge, no nos puede resultar demasiado útil a efectos practicos, ya que proporciona una tasa de SER demasiado elevada.

Algo parecido a lo comentado sobre LMS le ocurre a la alternativa no lineal RBF, ya que presenta exactamente la misma tendencia, aunque con una tasa de error menor.

El único de los algoritmos estudiados que presenta un resultado estable con el número de muestras de entrenamiento, es KNN, que tiene una sorprendente respuesta plana, produciendo incluso una menor SER al introducir la señal de variación. Sin duda es la mejor alternativa ante este tipo de señales.

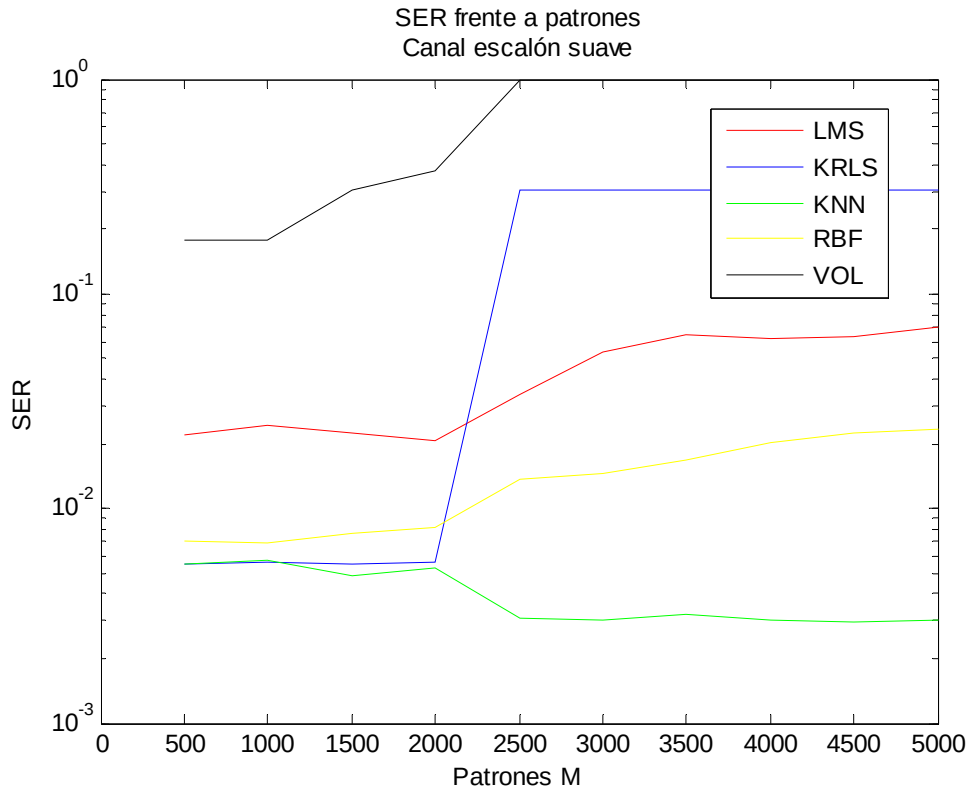



**Figura 5.34 - SER frente a patrones para canal escalón suave y entrenamiento dinámico guiado por decisión**

- Canal escalón medio: Los resultados obtenidos se pueden ver en la Figura 5.35.

En comparación con la variación escalón suave, vemos un empobrecimiento de las prestaciones al igual que ocurría para el caso dínamico.

El algoritmo de Volterra, ofrece una solución estable hasta que se introduce el entrenamiento guiado por decisión, siendo el punto en el que comienza la variación de coeficientes, cuando la solución que ofrece diverge. El resultado es semejante al caso escalón suave.

También obtenemos el mismo resultado para el algoritmo KRLS, que modifica la respuesta estable que nos ofrece tras introducir el entrenamiento guiado por decisión, a una solución equiprobable al introducir la variación escalón.

Un gran empeoramiento podemos ver en el comportamiento del algoritmo LMS, ya que no converge al introducir la señal de variación. Algo parecido le ocurre al algoritmo RBF, que tiende a una solución muy cercana a la unidad en términos de probabilidad de error.

Por último, tenemos que el algoritmo KNN es nuevamente el que mejor responde ante las altas cotas de variación que estamos tratando en este caso. Sufre un empeoramiento de una década en SER con respecto a la variación escalón suave, pero es el único algoritmo de todos los estudiados que no diverge ante esta señal de variación.

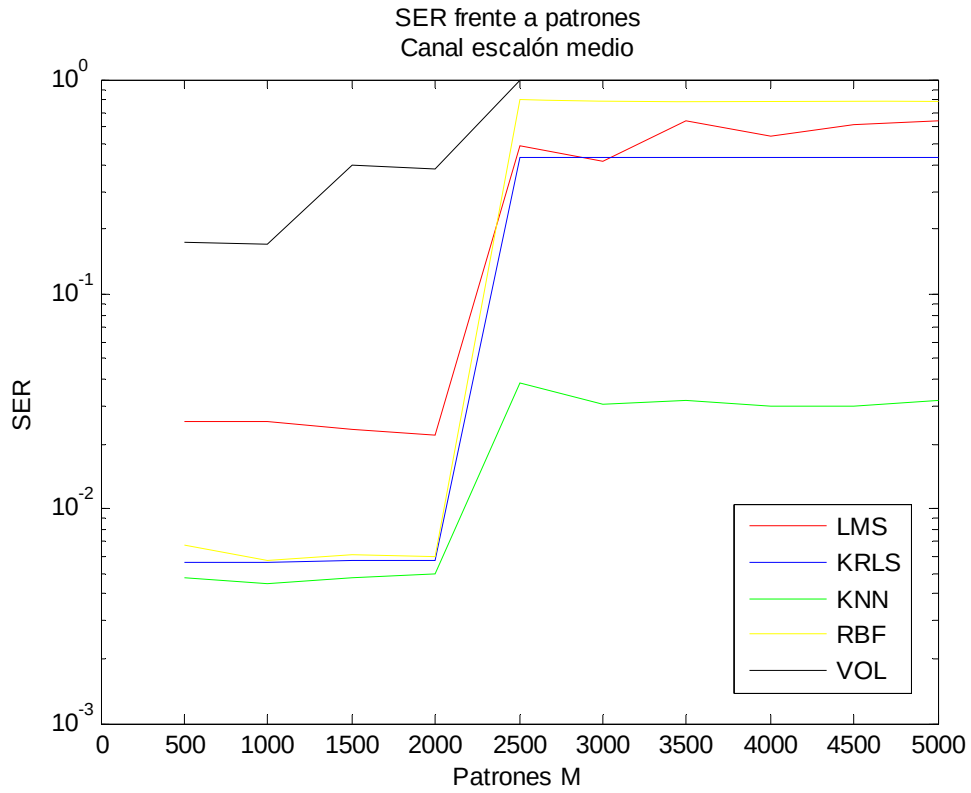



**Figura 5.35 - SER frente a patrones para canal escalón medio y entrenamiento dinámico guiado por decisión**

### 5.6.3.3 Gráfica error frente a patrones de entrenamiento

En este punto vamos a tener en cuenta las señales de error de los algoritmos, continuando con las que expusimos en el análisis estático, pero en este caso, sobre la duración del entrenamiento guiado por decisión. Con estos resultados queremos mostrar el comportamiento de los algoritmos ante la inclusión del nuevo modelo de entrenamiento, así como de la variación de los coeficientes de canal aplicada a ese nuevo entorno.

Para las gráficas que mostramos, y para que sirva a nivel comparativo con los resultados anteriores del presente proyecto fin de carrera, volvemos a emplear las variaciones de señal ya utilizadas.

Al igual que comentamos en las gráficas de error para el escenario estático, en nuestros resultados no vamos a representar el error del algoritmo

KNN, ya que al basarse en agrupamientos, el error obtenido toma valores $\pm 1$. Dicho error no tendría sentido en nuestras gráficas de error cuadrático medio.

Las gráficas que mostramos comienzan en la muestra 1000, ya que es donde introducimos el entrenamiento guiado por decisión (recordemos que las primeras muestras, hasta la 1000, correspondía con la fase de entrenamiento previo estático). Revisamos uno a uno cada uno de los escenarios empleados:

- Canal senoidal suave: Los resultados de las gráficas de error se pueden ver en la Tabla 5.16.

  Con respecto a KRLS, vemos que la cota de error que presenta es muy reducida, siendo el algoritmo que menor error ofrece de los mostrados en los resultados. Aparece igualmente una tendencia de error decreciente muy leve.

  Siguiendo a KRLS en niveles mínimos de error, tenemos a RBF, que ofrece un resultado muy parecido. Es evidente que ambos algoritmos se encuentran en una situación estable, no afectándoles ni la variación de canal, ni la introducción del entrenamiento guiado por decisión.

  Con respecto a LMS se observa el ajuste de error del algoritmo alcanzando sobre la muestra 2500 una situación estable. Es de mencionar que el algoritmo no introduce mayor error ante un cambio de canal, únicamente al iniciar el entrenamiento guiado.

  Por último, tenemos el algoritmo de Volterra que obtiene la mayor cota de error de todos los algoritmos que mostramos, tanto en valores máximos como en variación del ruido. Aparece una ligera tendencia decreciente en las primeras 500 muestras de la figura.

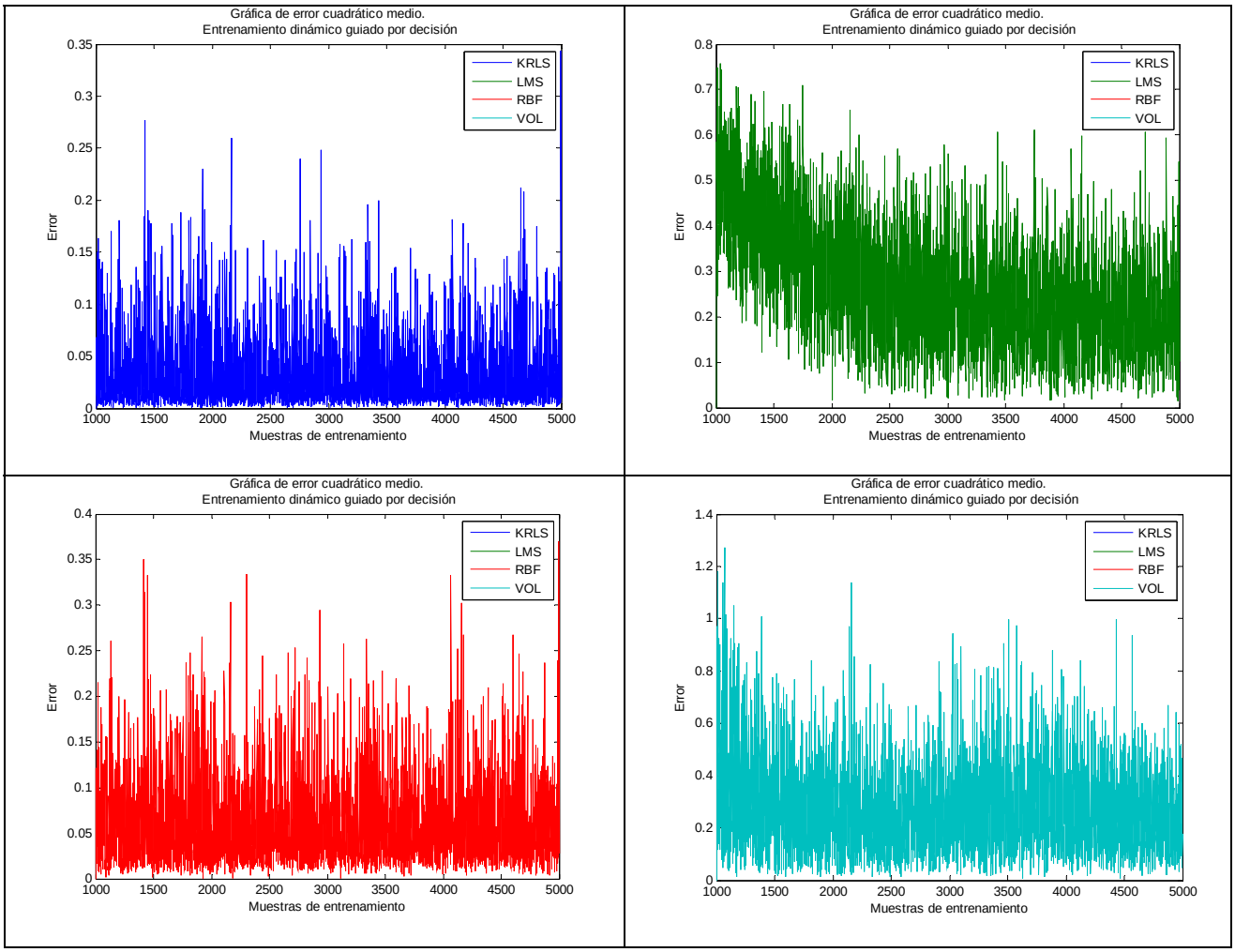



**Tabla 5.16 – Error cuadrático medio para canal senoidal suave y entrenamiento dinámico guiado por decisión.**

- Canal senoidal medio: Los resultados obtenidos aparecen en la Tabla 5.17.

Con respecto a la gráfica de KRLS, vemos como el error se ha duplicado en terminos medios, manteniendose a grandes rasgos la misma tendencia estacionaria que en el caso anterior.

El algoritmo LMS, tiene un comportamiento semejante al mencionado para el caso senoidal suave, con la salvedad de incrementar levemente el valor medio de su error. También se puede notar la oscilación del algoritmo a partir de la muestra 2000 y que llega a estabilizarse en la muestra 3500 ante la variación de los coeficientes de canal.

Con respecto a RBF, podemos observar fácilmente el incremento de error que supone la introducción de la variación dinámica de canal. A

pesar de eso, parece alcanzar una solución estable en torno a la muestra 4000 de nuestro entrenamiento.

Por último, tenemos el algoritmo de Volterra, que aparece semejante a la anterior señal de variación, aunque incrementando levemente los valores medios de error. También representa la variación de los coeficientes de canal con oscilaciones en el error, pero por el contrario en este caso, no parece obtenerse un estado estable.

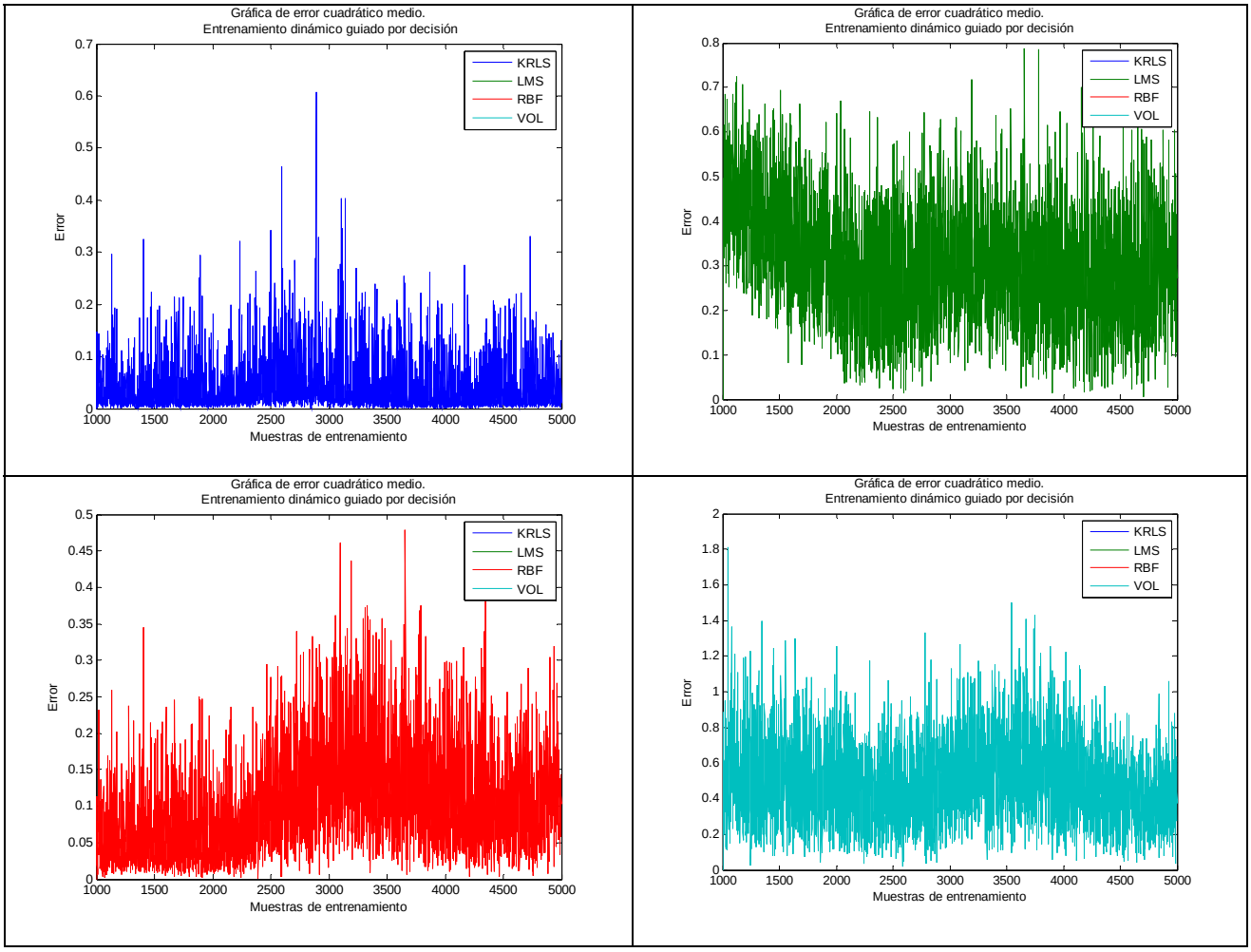



**Tabla 5.17 - Error cuadrático medio para canal senoidal medio y entrenamiento dinámico guiado por decisión.**

- Canal escalón suave: Los resultados obtenidos aparecen en la Tabla 5.18.

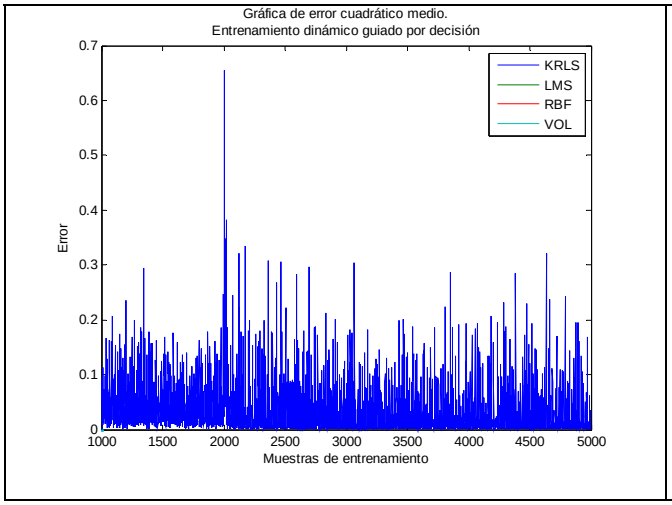
Antes de comenzar a comentar este caso, debemos mencionar que solo mostramos los resultados de KRLS, LMS y RBF, porque Volterra deja

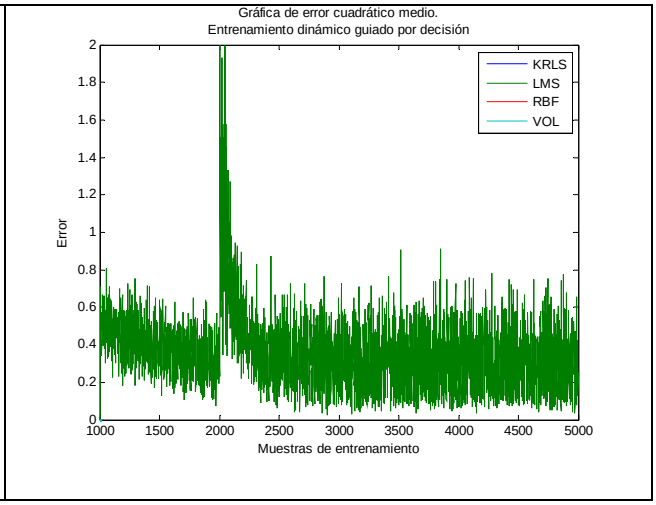
de converger, ofreciendo soluciones no reales. Esto viene reforzado por las oscilaciones que mostraba en su resultado de la Tabla 5.17.

En referencia a KRLS, vemos que las cotas de error son muy parecidas a las obtenidas con la señal senoidal media (este hecho es comprensible, ya que la tasa de variación de ambas señales es comparable). La única y principal diferencia con este caso, es la acentuación de error que aparece al introducir la señal dinámica en la muestra 2000. Se dispara el error del algoritmo, aunque rápidamente se vuelve a su valor constante inicial.

El algoritmo LMS, al igual que KRLS, presenta igualmente el mismo resultado que en senoidal media, con la salvedad del incremento del error en la muestra 2000, que en este caso es de mayor duración que en KRLS (en torno a 100 muestras). Una vez pasado este punto, el algoritmo vuelve a converger a una cota de error estable.

Por último, tenemos el algoritmo RBF, que vuelve a presentar un incremento de error muy elevado en la muestra 2000, para posteriormente volver a alcanzar un estado estable de error. Debemos tener en cuenta que es el peor algoritmo de los tres mostrados en tiempo de respuesta de convergencia.



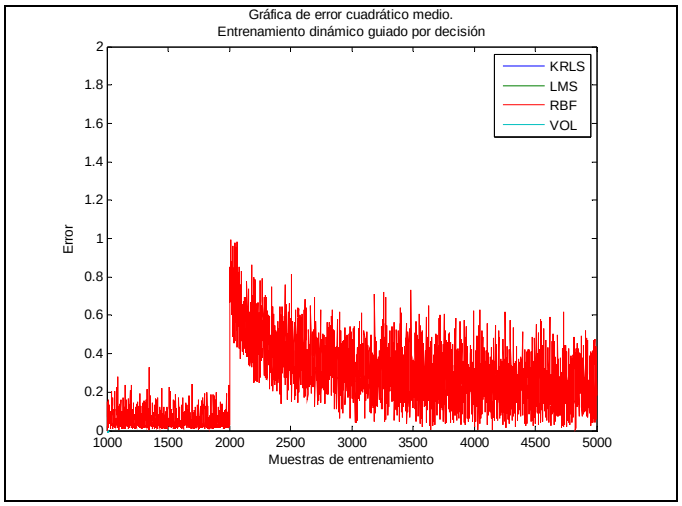



**Tabla 5.18 - Error cuadrático medio para canal escalón suave y entrenamiento dinámico guiado por decisión.**

- Canal escalón medio: Los resultados obtenidos de las simulaciones aparecen en la Tabla 5.19.

El resultado de KRLS es muy parecido al presente ante una señal escalón suave, con la diferencia de que en el punto de ruptura presente en la muestra 2000 alcanzamos una cota superior, y que a partir de ahí encontramos que la cota de error media ha crecido con respecto del punto previo a la variación.

El algoritmo LMS, presenta igualmente una media y desviación de error superior al caso anterior (en torno a la unidad). El algoritmo no es capaz de volver al punto inicial estático de error.

Por último, vemos como el algoritmo RBF presenta una gran variación debida a la introducción del entorno dinámico, consiguiendo saturar el algoritmo, y provocando un nuevo estado con un error muy elevado, llegando incluso a superar a la alternativa lineal.

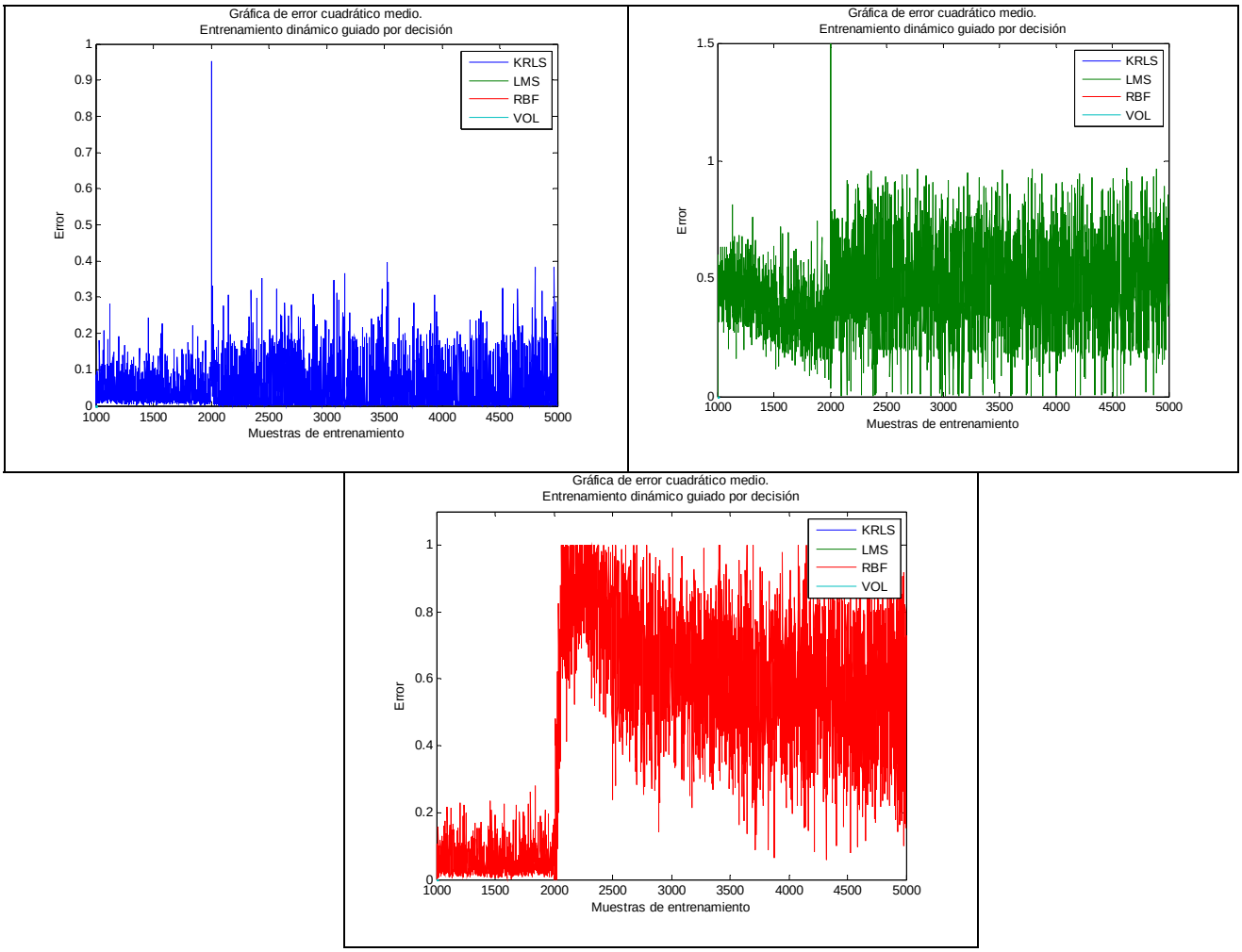



**Tabla 5.19 - Error cuadrático medio para canal escalón medio y entrenamiento dinámico guiado por decisión.**

### 5.6.3.4 Relación de eficiencia para entornos dinámicos guiados por decisión

Continuando con los resultados mostrados en el epígrafe 5.6.2.3, en el que mostramos los tiempos de ejecución de los algoritmos estudiados, así como el tamaño del diccionario de la máquina del algoritmo KRLS, vamos a volver a extraer resultados de eficiencia de nuestros algoritmos bajo este nuevo esquema de entrenamiento.

Al igual que hicimos para el entorno dinámico, hemos obtenido gráficas de tamaño de la máquina KRLS y tiempos de proceso desarrollados por los

algoritmos durante el proceso de entrenamiento. Todas ellas localizadas en el punto de mayor complejidad, correspondiente a los 0dB de SNR. Para corroborar esta afirmación, que también asumimos para el entorno estático y dinámico, hemos obtenido un resultado que nos muestra el tamaño del diccionario de la máquina del algoritmo KRLS a lo largo del barrido realizado en SNR, con lo cual justificaremos (al menos para KRLS) que la elección de dicho punto ha sido correcta.

#### *5.6.3.4.1 Gráfica de tamaño de máquina KRLS frente a SNR*

El resultado obtenido, presente en la Figura 5.36, nos demuestra que la afirmación realizada desde el epígrafe de entrenamiento estático y dinámico, que consistía en tomar el punto de SNR de 0dB como el caso peor (de nuestro barrido de 0 a 20 dB.) es cierta.

No obstante, tenemos que matizar que a la vista del resultado mostrado, tenemos que tener en cuenta que la afirmación es correcta para el algoritmo KRLS, que es el algoritmo principal de nuestro proyecto fin de carrera. Para el resto de algoritmos podemos extrapolar este resultado, junto con la propia definición de SNR y su tamaño de máquina constante, para considerarlo correcto igualmente. Este aspecto viene reforzado por las gráficas de tiempo del entrenamiento dinámico, y por las que mostraremos en este punto de entrenamiento guiado por decisión y que nos muestran como el comportamiento de los algoritmos de la comparativa, es semejante al mostrado por KRLS.

En la Figura 5.36, vemos igualmente que, aparte de que la complejidad del algoritmo decrece exponencialmente con el aumento de SNR, dicha complejidad aumenta en función de la señal de variación utilizada. Esta complejidad la entendemos como el número de elementos introducidos en el diccionario de KRLS.

Un aspecto a tener en cuenta es el caso de la señal de variación escalón media, en la que, tal y como hemos comentado anteriormente (5.6.2.1), los tiempos de procesamiento se hacen tan elevados que se convierten en inviables para simulaciones prácticas. Por esa razón, y al igual que se realizó en el epígrafe 5.6.2.1, hemos truncado el número de muestras de entrenamiento de 5000 a 2200

para obtener el resultado. A pesar de realizar esta reducción de muestras de entrenamiento, tenemos que la señal de variación escalón media es la más compleja de tratar para el algoritmo KRLS.

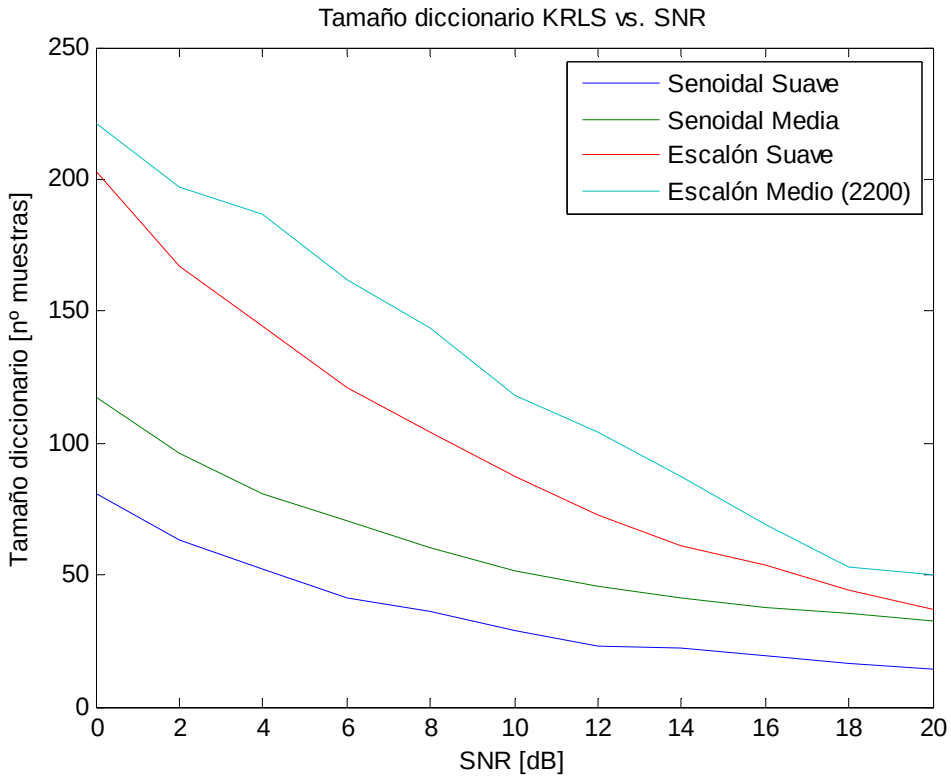



**Figura 5.36 – Tamaño del diccionario del algoritmo KRLS frente a SNR**

### *5.6.3.4.2 Gráfica de tiempos de procesamiento*

Al igual que realizamos para el entorno de entrenamiento dinámico, hemos vuelto a extraer los resultados correspondientes a los tiempos de procesamiento de todos los algoritmos bajo estudio.

De igual manera, hemos tenido que dividir los resultados en dos partes para cada una de las señales de variación, ya que el elevado valor de tiempo de KRLS, provoca que el resto de las gráficas no sean visibles. Por esa razón en la segunda gráfica de cada tabla, hemos centrado los resultados del resto de algoritmos a modo de comparativa.

Debemos destacar de nuevo la característica variable de estos resultados, que hacen que los resultados tengan significado relativo principalmente, por

encima del significado absoluto de los tiempos mostrados. También debemos considerar como resultado principal las líneas constantes de las gráficas, ya que los picos que aparecen se justifican principalmente con el mayor costo de procesamiento que supone la salida por pantalla de los resultados de las simulaciones.

Las simulaciones han sido realizadas sobre las mismas señales de variación que vamos utilizando durante todo el proyecto.

- Canal senoidal suave: El resultado obtenido aparece en la Tabla 5.20.

En ella podemos ver como KRLS proporciona tiempos de procesamiento muy elevados frente al resto de algoritmos, con la única salvedad de que dicho tiempo se mantiene constante durante todo el proceso de entrenamiento, y que no se ve afectado por la inclusión de la señal de variación.

Con respecto al resto de algoritmos, vemos que tienen un comportamiento muy estable igualmente, siendo LMS el que menor tiempo de procesamiento necesita. Esta última afirmación ya la habíamos supuesto, ya que es la única alternativa lineal de todas las mostradas. Los otros tres algoritmos: KNN, RBF y Volterra, presentan aproximadamente el mismo tiempo de procesamiento.

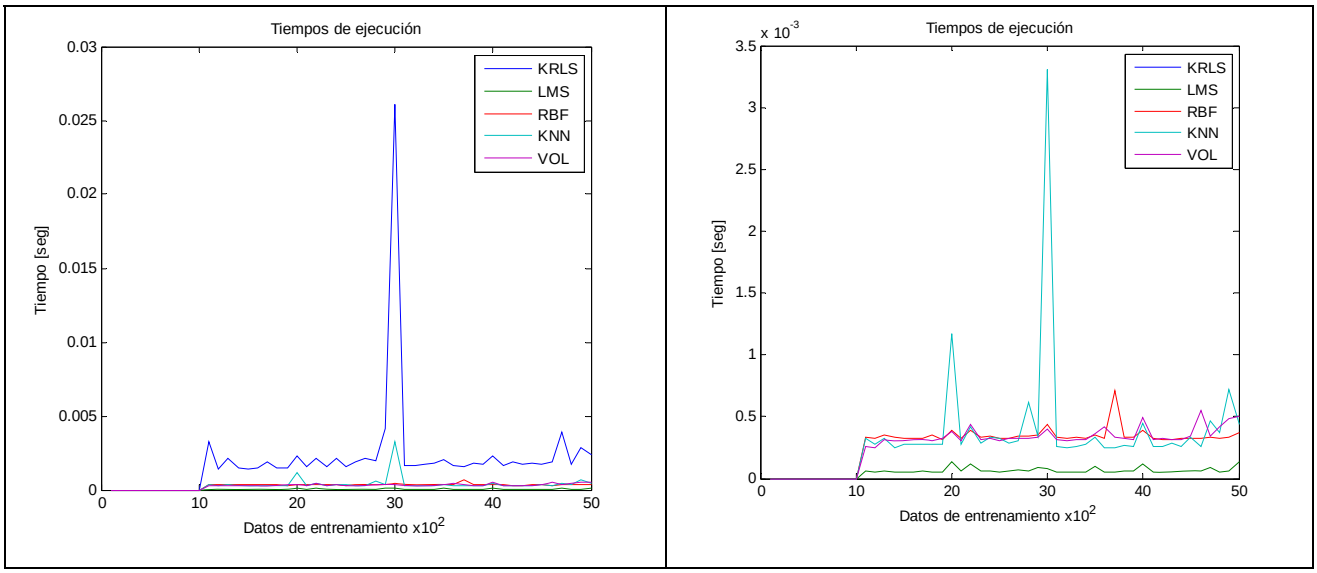



**Tabla 5.20 - Tiempos de ejecución para una señal de variación senoidal suave con SNR 0dB.**

- Canal senoidal medio: El resultado obtenido aparece en la Tabla 5.21.

En ella podemos ver de nuevo que el peor resultado de tiempos de procesamiento corresponde con la alternativa KRLS. Aparece una tendencia creciente en su curva de tiempos, principalmente derivada de su inclusión de nuevos elementos en su diccionario, tal y como veremos más adelante.

El resto de algoritmos no presentan ninguna pendiente, y siguen en un estado semejante al aparecido ante la señal senoidal suave.

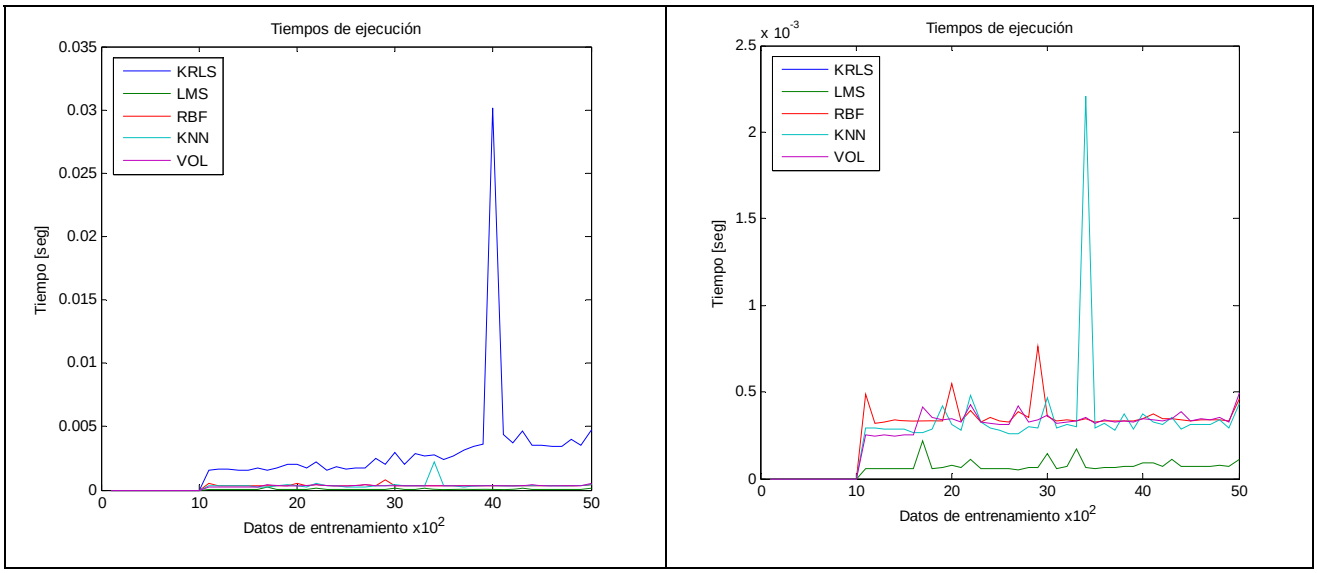



**Tabla 5.21 - Tiempos de ejecución para una señal de variación senoidal media con SNR 0dB.**

- Canal escalón suave: Los resultados que obtuvimos al introducir una señal escalón suave aparecen en la Tabla 5.22.

En este caso, volvemos a ver una pendiente en la curva de tiempos del algoritmo KRLS, aunque en esta ocasión se hace más evidente, comenzando desde la muestra 2000, donde introducimos la variación de canal.

El resto de algoritmos experimentan un pequeño incremento en sus tiempos de procesamiento, realmente irrisorios si los comparamos con el que presenta el algoritmo KRLS. A parte de esa pequeña pendiente, el resultado que muestran estos algoritmos es semejante al presente ante señales de variación senoidal.

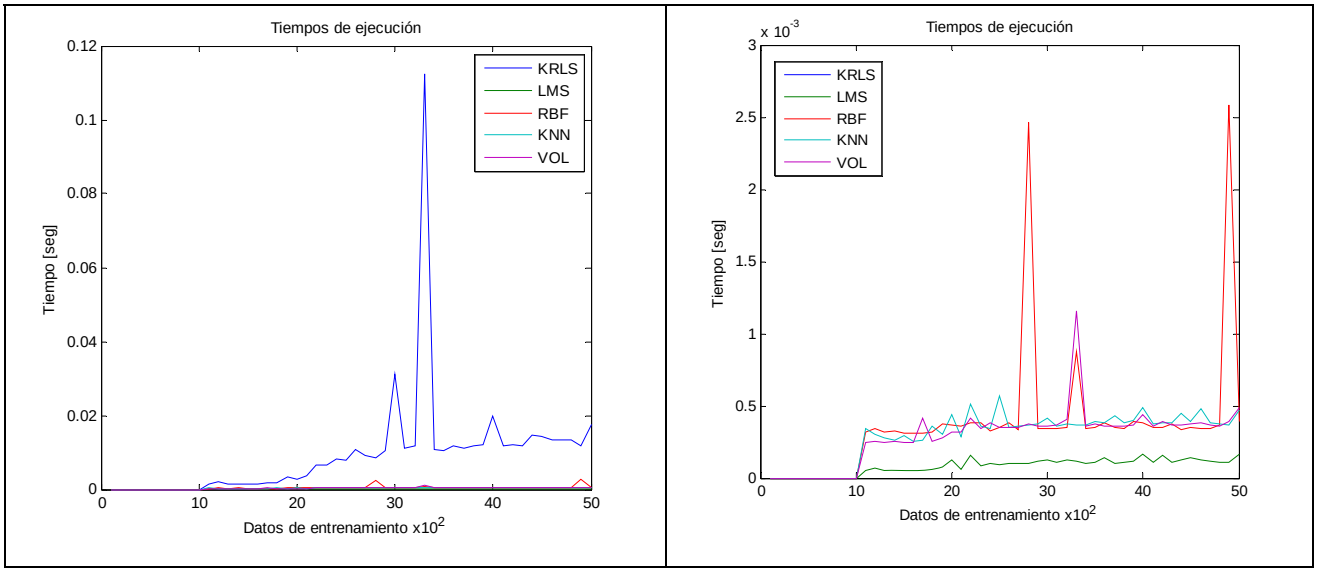



**Tabla 5.22 - Tiempos de ejecución para una señal de variación escalón suave con SNR 0dB.**

- Canal escalón medio: Los resultados obtenidos pueden ser vistos en la Tabla 5.23.

En este caso, tenemos que la pendiente del algoritmo KRLS ha aumentado considerablemente desde la muestra 2000, lo cual hace muy complicado su procesamiento muestra a muestra, debido a sus elevados tiempos de proceso. En este punto, el algoritmo KRLS es claramente deficiente.

Por otro lado tenemos que el resto de algoritmos se mantienen constantes devolviendo una gráfica semejante a la mostrada en el caso de la señal escalón suave.

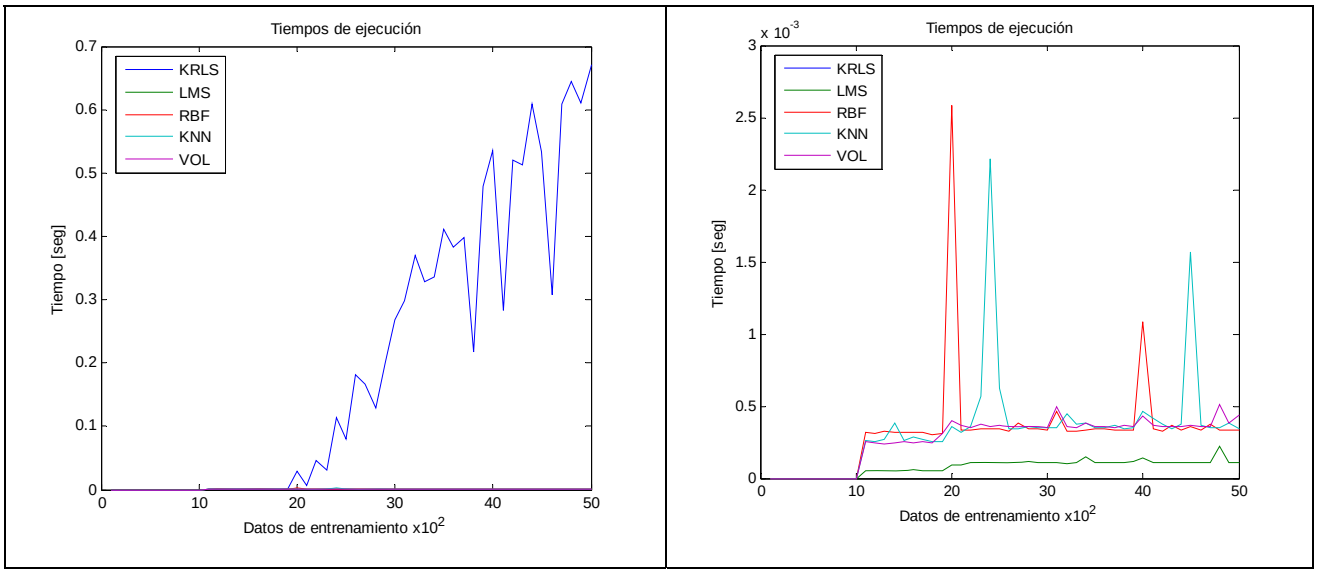



**Tabla 5.23 - Tiempos de ejecución para una señal de variación escalón media con SNR 0dB.**

### *5.6.3.4.3 Gráfica de tamaño de la máquina del algoritmo KRLS*

Al igual que obtuvimos para el entorno dinámico, hemos obtenido la evolución de los elementos en el diccionario del algoritmo KRLS durante el proceso de entrenamiento para las 4 señales de variación empleadas.

Como ya hemos comentado anteriormente, este apartado sólo muestra el tamaño de la máquina del algoritmo KRLS, porque consideramos que el resto tienen tamaño constante, ya que no crecen con el tiempo.

Los resultados de las gráficas que mostramos para la señal de variación escalón medio son las que justifican el truncamiento de datos presente en el epígrafe 5.6.3.1 del presente entrenamiento, ya que nos ayuda a demostrar el hecho de que el algoritmo KRLS presenta unos tiempos de procesamiento muy elevados.

Los resultados que hemos obtenido se encuentran presentes en la Tabla 5.24.

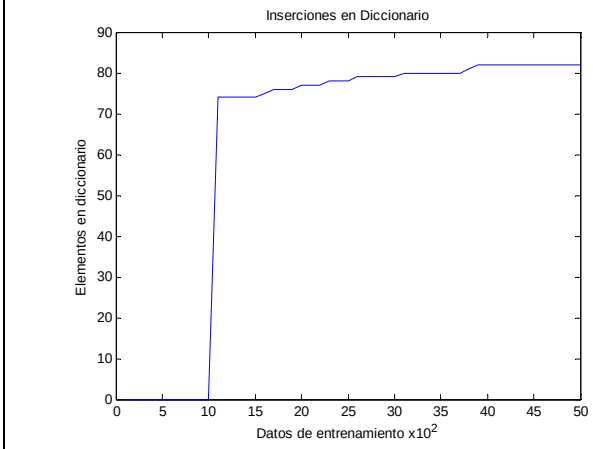
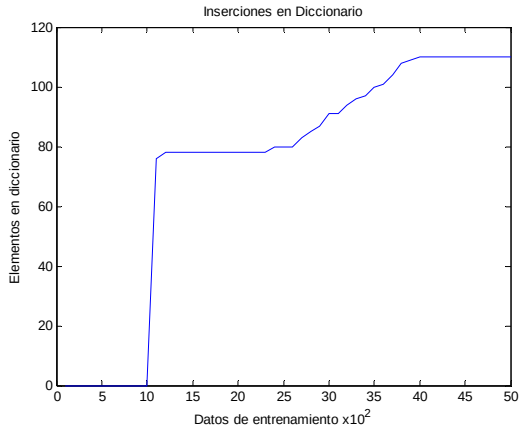
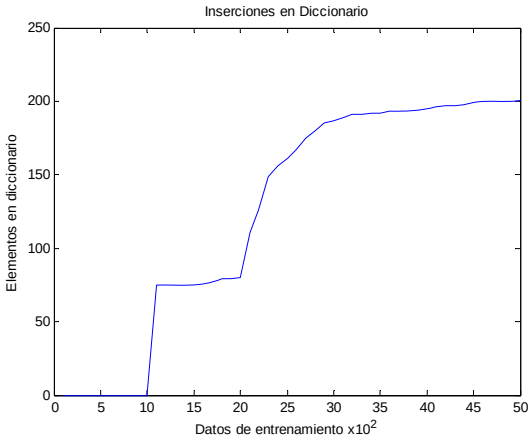
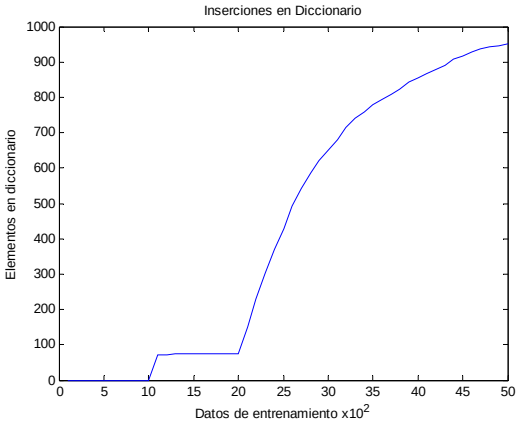
| Tipo | Variación: Suave | Variación: Media |
|---|---|---|
| Senoidal |   |   |
| Escalón |   |   |

**Tabla 5.24 – Inserciones en el diccionario del algoritmo KRLS durante el proceso de entrenamiento para SNR = 0dB.**

Estos resultados nos vuelven a reafirmar las ideas extraidas en el entrenamiento dinámico, en el que obtenemos un estado estable para el algoritmo KRLS ante señales senoidales, presentando a lo sumo una pendiente creciente lineal en las inserciones frente a las muestras de entrenamiento, y una pendiente exponencial para el caso de la señal escalón, derivando a una situación inestable para la señal escalón media, que nos lleva al truncamiento de los datos de entrenamiento en las gráficas de SER frente a SNR. Este hecho viene derivado de que al tener mayor número de elementos en el diccionario, tenemos una tasa de reducción mayor, y un tiempo de procesamiento mayor igualmente.

# Capítulo 6 Conclusiones

En este capítulo vamos a obtener las conclusiones globales de nuestro proyecto fin de carrera. Este epígrafe se ha dividido en 4 puntos: los tres primeros corresponden con los experimentos realizados, en los que mostraremos las conclusiones parciales de cada uno de ellos, y por otra parte, tendremos unas conclusiones finales de todo el proyecto fin de carrera. Nuestro objetivo es el de realizar estas conclusiones de manera incremental, comenzando por el primer caso estudiado, el entrenamiento estático. Posteriormente nos encargaremos del entrenamiento dinámico, apoyándonos en los resultados obtenidos en el entrenamiento estático. Después, mostraremos las conclusiones del último experimento desde el epígrafe de entrenamiento dinámico guiado por decisión, y para el que nos apoyaremos de los resultados obtenidos con el entrenamiento estático y dinámico. Por último tendremos las conclusiones finales del proyecto fin de carrera que recaeran sobre el último punto de este capítulo.

Todas las conclusiones que obtengamos en estos puntos, van a estar extraidas de los resultados obtenidos en el Capítulo 5, así como de la explicación teórica del Capítulo 4 y del estado del arte presente en Capítulo 3 principalmente y en el Capítulo 2 en menor medida.

## *6.1 Entrenamiento estático*

Para este punto vamos a basarnos en los resultados mostrados en el epígrafe 5.6.1.

Antes de empezar a comentar los resultados obtenidos, vamos a realizar un recorrido por las configuraciones establecidas para los parámetros de los

algoritmos estudiados. Lo realizamos en este punto, porque se fijan en el experimento estático, manteniendose constantes durante todo el proyecto fin de carrera. De forma común para todos los algoritmos, elegimos un retardo de igualación de 2 muestras, apoyándonos en el resultado que se muestra en la Tabla 5.4, donde relacionamos el resultado de nuestra implementación de KRLS con la obtenida por los autores del algoritmo, en función del retardo de igualación (según lo especifican en el documento de referencia [Y. Engel, S. Mannor, R. Meir;2004]).

Posteriormente definimos los parámetros a utilizar en nuestra implementación de KRLS, los cuales fueron diferentes a los elegidos por los autores, según especificamos en el epígrafe 5.2.2, donde realizamos un estudio sobre los retardos de igualación y parámetros posibles para KRLS. En este estudio ponemos de manifiesto que obtenemos unos resultados semejantes a los extraidos por los autores del documento cuando elegimos un kernel gaussiano con parámetro 'a' de valor 1 (corresponde con la varianza de la gaussiana), en lugar de un kernel polinómico de grado 8. Con respecto al valor del parámetro $\nu$, que como sabemos controla la comparación ALD del algoritmo, (y por tanto gobierna la inclusión de nuevas muestras en el diccionario) también resulta modificado, ya que nosotros establecemos un valor de 0.01 frente al valor 0.1 de los autores. Este cambio desemboca en que en nuestra implementación tendremos unas tasas de reducción del diccionario mucho menores que en la de los autores, lo que se traduce en que guardaremos menos elementos en el diccionario (a costa de aumentar levemente el error), con lo que reduciremos el tiempo de proceso del algoritmo.

Esta elección de parámetros diferente realizada para KRLS, ha sido documentada en el apartado 5.2.1 del presente proyecto. Aparte de los resultados correctos que obtenemos con nuestra elección de parámetros, no podríamos considerar erronea nuestra selección por ser diferente a la de los autores, ya que como ellos mismos especifican en el documento de referencia, muestran los resultados con kernel polinómico de grado 8 y $\nu$ con valor 0.1 sin haber realizado pruebas alternativas, lo cual nos deja paso libre a elegir cualquier alternativa a la suya con sólo justificar unas tasas de error semejantes.

Con respecto a la elección de parámetros para el algoritmo KNN, hemos establecido un compromiso entre tiempo de proceso y calidad de resultados del

mismo. Por esta razón decidimos elegir un valor de 3 vecinos para nuestro igualador no lineal y limitar el tamaño del mapa Knn de muestras etiquetadas a 200 muestras. La selección del primer parámetro se realizó para imponer una cota en tiempo de proceso por cada muestra nueva a catalogar, mientras que el tope en el tamaño del mapa, además de para limitar la complejidad, también tiene la función de evitar la propagación de errores para entornos dinámicos o con errores.

Para el algoritmo RBF, tomamos la misma varianza para los gaussianas de los centros que la de la fuente de datos. Por otro lado, los pesos del algoritmo competitivo (con $\eta$ premiamos al ganador y con $\gamma$ penalizamos al segundo), fueron elegidos de forma práctica con el objetivo de minimizar la SER ofrecida a la salida de la red RBF. Por último, tenemos la elección inicial de los centros $c_i$, que viene justificada para el canal que empleamos como estático y que aparece en la expresión 5.1, ya que tomamos como centros, las nubes de símbolos de salida de nuestro canal, tal y como aparecen en la Figura 5.4.

Con respecto a Volterra elegimos los parámetros conforme a unas gráficas de barrido realizadas en el apartado 5.5. Comentaremos en detalle su elección a continuación.

Comenzando con los resultados, tenemos la gráfica obtenida de SER frente al barrido en SNR que aparece en la Figura 5.14, en la que vemos que el algoritmo de Volterra presenta una pésima respuesta. Con este resultado podemos reforzar lo comentado en 5.5, donde pusimos de manifiesto que no conseguimos replicar los resultados mencionados por los creadores en su documento de referencia [C.-H. Tseng, E.J. Powers; 1993]. No obstante, bien es cierto, que en ese documento de referencia no se especifican valores exactos de resultado de simulaciones con el algoritmo, sino que la única conclusión que ofrecen es un conjunto de gráficas de nubes de puntos de una constelación 8-PSK. Dichas gráficas muestran un entorno ruidoso inicialmente y el resultado limpio de ruido tras introducir el algoritmo de Volterra de igualación de canal. Partiendo del hecho de que estos resultados ofrecidos no nos eran comparables, aceptamos el resultado de nuestra simulación de Volterra para los esquemas que hemos implementado, aún ofreciendo una respuesta claramente deficiente.

A modo de intentar optimizar el resultado obtenido con el algoritmo de Volterra, se modificó el algoritmo LMS empleado en su núcleo para estimar los kernel de Volterra que forman el equivalente del canal, por un algoritmo RLS. A pesar de este cambio, no conseguimos mejorar los resultados por lo que volvimos al algoritmo LMS inicial. Por último realizamos un barrido en los parámetros que gobiernan el funcionamiento del algoritmo, obteniendo los resultados que aparecen en la Figura 5.12 para el caso estático. Dicha gráfica no dejaba lugar a dudas, ya que observando un gran abanico de valores del número de bloques y de $\beta$, veiamos que el comportamiento del algoritmo era más cercano a la divergencia que a una solución razonable. En particular aparecían tres regiones claramente diferenciadas: por un lado teníamos que una parte importante del barrido correspondía con que el algoritmo divergía a valores infinitos, lo cual nos era inservible. El segundo grupo correspondía con valores de SER equiprobables, lo cual nos era inservible igualmente. Por último teníamos una zona bastante estrecha, que nos ofrecía la menor tasa de error en el entrenamiento estático realizado. Esos valores fueron los que se han mantenido durante todas las gráficas de este proyecto fin de carrera, y nos dan una idea de las claras deficiencias que presenta este algoritmo.

Una posible justificación al comportamiento de Volterra puede venir desde el punto de partida de su diseño como igualador, ya que emplea la teoría de punto fijo para encontrar una solución al problema. Matemáticamente se encuentra reforzado, pero prácticamente resulta en muchos problemas, ya que son precisos determinados valores de $\beta$ para que se produzca un mapeo de contracción que garantice la existencia del punto fijo que nos proporcione nuestra solución correcta. Bajo nuestro punto de vista, ese intervalo tan estrecho de valores de $\beta$ que responden a ese requisito, a nuestro juicio, es lo que hace que el algoritmo tenga la respuesta de barrido que aparece en la Figura 5.12.

Continuando con la Figura 5.14, vemos la poca diferencia existente entre los algoritmos KRLS, KNN, RBF y LMS hasta aproximadamente los 10 dB. En especial, resulta significativo que la alternativa lineal LMS iguale el comportamiento del resto de algoritmos no lineales (mejorando incluso el del algoritmo no lineal de Volterra), ya que si echamos mano de los resultados obtenidos de tiempo de ejecución de los algoritmos en la Tabla 5.7 y Tabla 5.8 para SNR 15dB y 0dB respectivamente, es el que menor complejidad representa

(si asemejamos tiempo de proceso con complejidad del algoritmo, lo cual parece ser una relación coherente). Todo esto ocurre con una no linealidad cúbica en la definición de nuestro canal de ejemplo, tal y como lo especifican los autores del documento de referencia de KRLS (expresión 5.1), lo cual da más valor al resultado del algoritmo LMS, ya que no puede construir una frontera de decisión no lineal frente al resto de contricantes.

A partir de ese punto de ruptura colocado en 10 dB podemos ver como los algoritmos no lineales se desmarcan claramente de la alternativa lineal, aventajandole en 2 décadas aproximadamente. Otro punto a tener en cuenta acerca de la gráfica presente en la Figura 5.14, es el resultado tan parecido que obtenemos con las otras tres alternativas no lineales durante todo el barrido en SNR. Esto nos ha ayudado ha especificar este resultado como punto de partida común para los tres algoritmos, y sobre el que continuaremos introduciendo las variaciones de señal en los coeficientes de canal. En definitiva y como conclusión de la Figura 5.14, debemos decir que el resultado que ofrecen KRLS, RBF y KNN es bueno en terminos de SER bajo un esquema de igualación no lineal.

Esta última valoración la apoyamos en el resultado numérico que nos ofrecen los autores del documento de referencia de KRLS para el mismo entorno de igualación no lineal que planteamos en este módulo.

Nuestras conclusiones vienen reforzadas por los resultados mostrados en la Figura 5.15, en donde realizamos un barrido sobre el número de muestras del proceso de entrenamiento para una SNR de 15 dB. En esa gráfica vemos que la estabilidad en los algoritmos es muy elevada, enganchandose a la solución correcta nada más comenzar con el entrenamiento. Esto nos viene a asentar de nuevo el estado común estacionario de partida de todos los algoritmos, y que deja paso libre al resto de esquemas de aprendizaje que se han planteado.

Otro resultado alternativo que vuelve a mostrar la estabilidad de los algoritmos en este entrenamiento estático, es la Figura 5.17, en donde podemos ver las gráficas de error cuadrático medio a lo largo de todo el proceso de entrenamiento para una SNR de 15 dB. En esa gráfica se puede ver que el algoritmo lineal LMS es el que mayor tiempo necesita para converger, lo cual es algo que era de esperar al enfrentarse a un canal no lineal. Entre las alternativas

de KRLS y RBF que obtenían unos resultados muy parecidos en la Figura 5.14, vemos que si presentan diferencias en sus errores ya que el algoritmo KRLS presenta un error más alto inicialmente, pero con un tiempo de convergencia mucho mas reducido que RBF. Esto permitirá a KRLS adaptarse a nuevas señales más rápidamente si el cambio no hace disparar el error a cotas que hagan diverger al algoritmo.

Otro aspecto por comentar de los resultados obtenidos, aunque únicamente añade valor para comprender gráficamente los resultados mostrados, son las gráficas de contorno que se muestran en la Figura 5.16. En ellas destaca la curva producida por el algoritmo LMS, en contraste con las fronteras de decisión no lineales que tienen el resto de algoritmos. Estas curvas también nos vienen a certificar de nuevo el mal comportamiento de nuestra implementación del algoritmo basado en kernel de Volterra, ya que aparece con una frontera de decisión no lineal, pero mal colocada con simetría par, haciendo que no pueda dividir correctamente las nubes de puntos. También es relevante que la curva que mejor se adapta a los datos es la correspondiente a KNN, con aspecto ruidoso, debido a que es un algoritmo basado en agrupamientos. Por otro lado tenemos la curva de KRLS, que separa las nubes de puntos correctamente, pero incluye unas nuevas fronteras en los exteriores que podrían provocar errores en la estimación ante una varianza de la fuente de datos de entrada elevada. Por último tenemos la curva de RBF que es la curva no lineal con menor grado, lo cual le hace incurrir en un mayor error que KRLS y KNN como se muestra en la Figura 5.14, aunque dicho error sea muy reducido en este caso(se incrementa con un canal variante como ya hemos visto).

Nos resta comentar un último aspecto de los resultados obtenidos, correspondiente al cálculo de complejidad de los algoritmos reflejado en la medición del tiempo de entrenamiento, tal y como se muestra en la Tabla 5.7 y en la Tabla 5.8 para una SNR de 15dB y 0dB. En esas tablas, vemos que la mayor complejidad corresponde al algoritmo KRLS, lo cual es un contrapunto a los buenos resultados que ha mostrado. Esta complejidad viene derivada de la necesidad de aplicar la función de kernel con todos los elementos del diccionario por cada muestra nueva que introducimos en el entrenamiento. Esta tarea es costosa y crece cuanto mayor número de elementos tengamos almacenados en el diccionario. El algoritmo RBF gana la partida de complejidad a KRLS, ya que

reduce a la mitad su tiempo de procesamiento. Otro punto en contra de KRLS es que aumenta su tiempo de proceso ante una reducción de SNR, lo cual lo convierte en un algoritmo de complejidad variable, frente al resto considerados más constantes.

En este resultado de complejidad, tenemos que comentar el erroneo valor que se muestra para el algoritmo KNN, ya que el entrenamiento de dicho algoritmo radica principalmente en introducir elementos etiquetados previamente en el mapa Knn. Por esta razón, no tiene ninguna complejidad asociada, a excepción del proceso de obtención de SER, donde se calculan las distancias de todo el mapa Knn con cada muestra de la trama de test.

Por otro lado, tenemos que el algoritmo LMS es el que mejor complejidad muestra, lo cual era de esperar, ya que es la única alternativa lineal, y fue incluido en la comparativa, por su sencillez.

Como conclusión de este esquema estático, tenemos que apoyándonos en la Figura 5.14, podemos decir que la mejor opción hasta el punto de ruptura de 10 dB de SNR, la proporciona la alternativa lineal LMS, tanto en SER, como en eficiencia. A partir de ese punto de 10 dB, tendríamos las alternativas de KRLS, KNN y RBF muy similares. Entre estos tres, nos decantaríamos por RBF o KNN por términos de eficiencia, aunque se debería garantizar la existencia de suficientes muestras de entrenamiento para que el algoritmo RBF pueda converger hacia la solución correcta, ya que posee un tiempo de convergencia de aproximadamente 400 muestras.

## *6.2 Entrenamiento dinámico*

Para comentar este punto nos apoyaremos en los resultados que hemos obtenido en el epígrafe 5.6.2 de nuestros resultados experimentales.

Continuamos con nuestra disertación acerca de la conveniencia de los resultados del algoritmo de Volterra, realizada en las conclusiones sobre el entrenamiento estático. Si nos encontrabamos ante un problema en nuestro

entorno estático ante la reducida zona de barrido de parámetros donde obteniamos una respuesta coherente de nuestro algoritmo, en esta ocasión empeoramos dicha situación, tal y como se puede ver en la Figura 5.13. En esta figura volvimos a realizar el barrido de parámetros ($\beta$ y número de bloques) sobre un entorno dinámico de prueba con una variación senoidal suave (La definición de estas señales se puede ver en la Tabla 5.9 y la comentaremos más adelante). El resultado obtenido en comparación con el que tuvimos para el caso estático, fue que la región estable se reducía aún mas, manteniendose aproximadamente los mismos valores óptimos de $\beta$ y del número de bloques. Otro aspecto que resultaba alarmante fue el incremento de la región de inestabilidad del algoritmo, eliminando en cierta medida el área de equiprobabilidad. Esto nos dice que el algoritmo de Volterra modifica el valor óptimo de sus parámetros en función de la señal de variación introducida, y que empeora su respuesta considerablemente con la inclusión de un entorno dinámico.

Para comentar las gráficas obtenidas en nuestro entorno dinámico, debemos comentar previamente las señales genéricas que hemos empleado para introducir la variación de los coeficientes. Dichas señales aparecen en la Tabla 5.9 y sus tasas de variación en la Tabla 5.10. En ellas basamos nuestros experimentos, con el objetivo de comprobar el enganche de los algoritmos ante señales de variación lenta, como puede ser la senoide, y la capacidad de aprendizaje ante un nuevo canal, como puede ser un escalón. Para cada una de las señales establecemos dos niveles: suave y medio, con incremento de una década aproximada en su variabilidad, con el objetivo de estudiar el comportamiento de los algoritmos ante diferentes tasas de variación.

Se ha garantizado la comparación de resultados entre el entorno estático y el dinámico, ya que se siguen empleando las mismas gráficas de resultado: SER frente a SNR, SER frente a patrones de entrenamiento y gráficas de eficiencia. Igualmente utilizamos los mismos parámetros que se definieron en los experimentos estáticos.

Apoyándonos en esas señales, es como hemos construido los resultados de este punto. Viendo las gráficas dinámicas de SER frente a SNR que aparecen en la Figura 5.18 para una variación suave, podemos certificar que la señal de variación no afecta significativamente a los algoritmos, ya que se obtienen los

mismos resultados que para el caso estático. Esto viene reforzado por la gráfica de SER frente a patrones obtenida igualmente para una señal senoidal suave y que aparece en la Figura 5.24. En conclusión, desde el punto de vista de esta variación se mantienen las conclusiones que se establecieron para el entorno estático.

Examinando las gráficas, tanto de SNR como de patrones, obtenidas para la señal senoidal media, que aparecen en la Figura 5.19 y la Figura 5.25, vemos que el comportamiento de enganche de los algoritmos a la señal senoidal cambia con respecto a los resultados del entorno estático. En este caso, se observa que en la gráfica de SER frente a SNR, obtenemos una variabilidad considerable del algoritmo KNN. Este hecho unido a que el resultado de KNN que se muestra en la gráfica de patrones aparece muy variable y no concuerda con la tasa de error presente en la gráfica de SNR, hace que el algoritmo KNN produzca diferentes respuestas en función de la ejecución que se realice, probablemente determinadas por los elementos disponibles en el mapa Knn (si los elementos del mapa son erroneos, el algoritmo KNN estará predispuesto a propagación de errores). Esta variabilidad hace que el algoritmo sea inviable para la señal senoidal que nos ocupa.

Para esta señal, vemos igualmente que el resultado de Volterra es igualmente inservible, ya que sus resultados son equiprobables, haciendo que no sirva para el problema de igualación de canal no lineal que nos ocupa.

Un aspecto muy importante que podemos ver en las gráficas es el nuevo resurgimiento del algoritmo LMS, que modifica el punto de ruptura de los 10 dB presentes en el entorno estático a los 15 dB aproximadamente. Por último tenemos el resultado de RBF y KRLS que son semejantes durante todo el proceso de entrenamiento.

Como conclusión de esta señal de variación, tomaríamos el algoritmo LMS hasta los 15 dB, ya que es más eficiente y obtiene tasas de SER semejantes a las variantes no lineales. A partir de este punto, podríamos tomar tanto RBF como KRLS, aunque basándonos en el estudio de eficiencia realizado en el entorno estático, ganaría por eficiencia RBF siempre y cuando se garantizase un tiempo de entrenamiento suficiente que le permitiera engancharse a la nueva señal.

Para examinar la capacidad de adaptación ante un nuevo entorno de los algoritmos, introducimos la señal de variación escalón en su variante suave inicialmente, apareciendo los resultados obtenidos de barrido en SNR en la Figura 5.20 y de barrido en patrones en la Figura 5.26. En ellas podemos ver que el algoritmo de Volterra sigue sin converger, y que el comportamiento de los algoritmos es bastante peor al encontrado ante la señal de variación senoidal, a excepción de KNN. El algoritmo LMS parece no encontrar la solución correcta a la variación, y algo parecido le ocurre al algoritmo RBF, principalmente, porque no puede adaptar los centros de sus gaussianas a las nuevas nubes de puntos presentes a la salida del canal. Este comportamiento de RBF es derivado, a nuestro juicio, de su mayor tiempo de convergencia, tal y como mostramos en el esquema estático, ya que le impide adaptarse a una nueva señal rápidamente.

Al contrario de lo que le pasa a RBF, tenemos que KRLS (presentaba un tiempo de convergencia bastante menor) si parece adaptarse a la nueva señal, provocando que a altos valores de SNR tenga aproximadamente el mismo valor de SER que KNN. No obstante aparece un problema con el algoritmo KRLS, ya que incrementa sus tiempos de proceso de manera exponencial. Ello es debido a que las inserciones al diccionario se disparan ante el cambio de señal, tal y como hemos visto en la gráfica de inserciones presente en la Figura 5.21. Para este caso particular de señal de variación, este incremento de tiempo de procesamiento es asumible, pero en el caso de la señal escalón medio presenta un incremento demasiado elevado, (aparece en la Figura 5.23) que hace que el algoritmo KRLS no sea viable en ese caso. En esta situación, el algoritmo KRLS es muy dependiente de la señal de variación introducida y en particular de la tasa de variación que tenga. Este problema tendría solución si el algoritmo KRLS permitiera olvidar del diccionario las muestras inservibles del antiguo canal, que solamente añaden error a las estimaciones en presencia de un nuevo canal.

Para la variación escalón suave elegiríamos como algoritmo más viable a KRLS, ya que aunque KNN tiene la mejor respuesta para SNR mayor que 10 dB, presenta una irregularidades bastante importantes en bajas SNR. Por el contrario, el algoritmo KRLS mantiene constante la curva durante todo el barrido de SNR y de patrones.

El último paso que realizamos fue el de utilizar la señal escalón medio, la cual tiene una gran tasa de incremento en los coeficientes del canal. Los resultados que obtuvimos aparecen en la Figura 5.22, y en la Figura 5.27. En la gráfica de SER frente a SNR se nos volvió a presentar el problema de la convergencia del algoritmo KRLS en relación a las inserciones en el diccionario. Por esta razón se redujo el número de muestras de la gráfica, ya que el tiempo de proceso del algoritmo era inviable. Este es el motivo por el que no coinciden los valores para KRLS en las gráficas de SNR y patrones, ya que tuvimos que truncar la longitud de entrenamiento en el resultado de SNR, mientras que la gráfica de patrones sigue manteniendo la longitud original aplicada al resto de gráficas de patrones del proyecto. Esta modificación realizada en la longitud de entrenamiento no afecta al resto de algoritmos, por lo que el resultado sigue siendo válido. El único afectado es KRLS, pero no demasiado, ya que el algoritmo queda completamente descartado para esta señal de variación al ofrecer un tiempo de proceso tan elevado.

Con respecto al resto de algoritmos, tenemos que al igual que ocurría en la señal senoidal media, tenemos que la alternativa LMS gobierna el resultado hasta el punto de ruptura de 10 dB. A partir de ahí tenemos que el único algoritmo que supera al algoritmo lineal es KNN. En la gráfica de patrones, se puede observar como afecta el cambio de señal a los resultados, provocando en media un incremento de una década en SER, irrecuperable durante el resto del entrenamiento.

Basándonos en los resultados de eficiencia obtenidos en las tablas: Tabla 5.11,Tabla 5.12, Tabla 5.13 y Tabla 5.14, para el peor caso de SNR (0 dB) llegamos a la conclusión de que los tiempos de procesamiento del algoritmo KRLS son con gran diferencia los más elevados de todos los algoritmos bajo estudio. Otro punto en su contra es la excesiva variabilidad de esos tiempos de procesamiento con respecto a la señal de variación introducida, con respecto al resto de algoritmos, que se mantienen prácticamente constantes. Entre el resto de algoritmos destaca los mínimos tiempos de proceso de LMS, que únicamente aumentan para la señal escalón, y el excelente resultado en terminos de complejidad y estabilidad de los algoritmos RBF y Volterra. Por otro lado tenemos que el algoritmo KNN presenta el doble tiempo de proceso que RBF y Volterra, con la misma variabilidad del algoritmo LMS.

Este comportamiento del algoritmo KRLS, viene reforzado por las gráficas obtenidas en la Tabla 5.15, en las que observamos el tamaño de la máquina del algoritmo, entendiéndolo como el número de elementos introducidos en el diccionario, frente a las muestras de entrenamiento. En dichas gráficas, vemos como aumenta el tamaño del diccionario para una señal senoidal, alcanzando un estado estable con una pendiente lineal, y con una pendiente exponencial para el caso de la señal de variación escalón, que lleva a la situación de no encontrar una situación estable durante las 5000 muestras de entrenamiento en el caso medio. Esto produce el problema que experimentamos en nuestras gráficas de SER frente a SNR ante un escalón medio, en el que las inserciones en el diccionario son tan elevadas, que los tiempos de proceso son impracticables. En estos resultados no incluimos el tamaño del resto de máquinas, al considerarlas constantes, basándonos en su definición teórica y en resultados prácticos no mostrados.

Como conclusión del entrenamiento dinámico, debemos mencionar que la mejor alternativa ante todas las señales la presenta el algoritmo LMS hasta la cota aproximada de 10 dB en un compromiso de eficiencia contra tasa de error. A partir de ese punto, nos encontramos con que KRLS ofrece buenas capacidades de enganche (examinadas con la señal senoidal), al contrario de KNN, y aceptables resultados en adaptación ante un nuevo canal, en los que KNN sale victorioso, y RBF como claro perdedor (en enganche obtuvimos buenos resultados con RBF). Como media de ambas características de adaptación, elegiríamos el algoritmo KRLS, aunque mejoraría claramente sus prestaciones y su tiempo de proceso, si permitiera eliminar entradas inservibles de su diccionario ante un cambio total del canal. No obstante, debemos tener en cuenta la problemática mostrada sobre la eficiencia de KRLS, por lo que sólo se podrá tomar como mejor opción ante valores de SNR y señales de variación controlados, ya que sino los tiempos de procesamiento se dispararían. Por esta razón no podemos determinar un algoritmo como claro ganador, ya que KRLS presenta la deficiencia en tiempo de proceso, RBF en situaciones de cambio brusco del canal, y KNN en situaciones de seguimiento de los coeficientes.

## *6.3 Entrenamiento guiado por decisión*

Para los resultados obtenidos con el esquema de aprendizaje guiado por decisión, que comentamos en el punto 5.6.3, nos hemos basado en los resultados obtenidos al aplicar de nuevo las mismas señales de variación detalladas en la Tabla 5.9. Para estas simulaciones volvimos a repetir los mismos parámetros de configuración que para el resto de experimentos.

Este esquema de entrenamiento es el más importante de los tres mostrados, ya que es el que consideramos como definitivo en el proyecto. Los otros dos, los podemos entender como pasos intermedios para llegar al entrenamiento guiado por decisión y que nos ayudan a dar una visión más global del comportamiento de los algoritmos.

El primer paso fue introducir la señal de variación senoidal suave, en la que obtuvimos los resultados de la Figura 5.28 para la gráfica de SER, la Figura 5.32 para patrones y la Tabla 5.16 para las gráficas de error. Examinando esos datos, habíamos llegado a la conclusión de que el algoritmo de Volterra seguía sin ofrecer una alternativa viable y convergente, durante todo el barrido de SER. Este aspecto no ha hecho nada más que repetirse durante todos los esquemas de aprendizaje, y en particular ante canales variantes. Al igual que ocurría para el caso dinámico, tenemos que el algoritmo LMS presenta una alternativa a tomar en cuenta para valores de SNR menores a 10 dB. A partir de esa cota toman el mando las alternativas lineales como KRLS, KNN y RBF, con resultados semejantes. No obstante, tenemos que KNN presenta alguna variabilidad a valores de SNR bajos, por lo que se podría descartar como elección óptima. Examinando la gráfica de patrones, vemos que el único algoritmo que ha experimentado problemas ante el cambio de esquema estático a guiado por decisión, ha sido el algoritmo de Volterra, el cual pasa a diverger.

Como mejores opciones para este intervalo, nos valdría tanto RBF como KRLS, aunque elegimos RBF por presentar una menor complejidad reflejada en tiempo de proceso.

El siguiente paso que realizamos para analizar la capacidad de seguimiento de los algoritmos, fue la introducción de la señal senoidal media, incrementando de esta manera la tasa de variabilidad de los coeficientes de canal.

Al igual que ocurría en el entorno dinámico, queda patente la gran variabilidad sufrida por el algoritmo KNN, que llega hasta los límites de devolvernos diferente valor de SER en la gráfica de SNR que en la gráfica de patrones a 15dB (aparece en la Figura 5.33). Esto no es más que una muestra de nuevo del pobre rendimiento que devuelve el resultado ante situaciones de seguimiento de la señal, tal y como se puede ver en la Figura 5.29, donde aparece la gráfica de SER frente al barrido en SNR. En general, los resultados que obtenemos ante esta señal son semejantes a los obtenidos para el entorno dinámico, por lo que volvemos a tener como mejor opción el algoritmo LMS hasta la cota de 12 dB. A partir de ese punto gobierna la gráfica tanto KRLS como RBF, siendo en cualquier caso RBF el que presenta menor complejidad. Observando las gráficas de error obtenidas en la Tabla 5.17 para 15 dB, vemos la oscilación presente en el algoritmo LMS, y el incremento de error en RBF ante el cambio de señal, frente al valor constante de KRLS. A pesar de eso, al presentar una menor complejidad seguiríamos optando por RBF como la mejor opción en esta señal, seguida muy de cerca por KRLS en el problema de seguimiento de señal.

Para examinar el comportamiento de los algoritmos ante un nuevo canal, introducimos la señal escalón suave. Los resultados que obtuvimos fueron mostrados en la Figura 5.20 para la gráfica de SER frente a SNR, la Figura 5.34 donde se muestra el resultado en barrido de patrones y por ultimo la Tabla 5.18 para los resultados de error cuadrático medio.

Observando dichas gráficas llegamos a la conclusión de la gran diferencia existente con las obtenidas para el entorno dinámico, principalmente sobre dos de los algoritmos bajo estudio: Volterra y KRLS. Con respecto a Volterra vemos que diverge a valores infinitos ante la entrada de la nueva señal. Este hecho no resulta demasiado relevante, ya que los resultados obtenidos para el algoritmo eran muy deficientes. Por el contrario, si es sorprendente, el empeoramiento del algoritmo KRLS, ya que pasa de considerarse la segunda mejor opción para este tipo de problemas por detrás del algoritmo de agrupamiento KNN a obtener una respuesta equiprobable por detrás del resultado devuelto por la alternativa lineal. Este

comportamiento puede ser motivado en gran parte por la deficiencia que ya comentamos en el apartado dinámico, y que radicaba en el almacenamiento en el diccionario de muestras del antiguo estado del canal, y que interfieren en la adaptación al nuevo estado. SI unimos este aspecto con el esquema de entrenamiento por decisión tenemos todos los ingredientes necesarios para obtener un caso claro de propagación de errores como es la situación que nos ocupa. La solución que seguimos planteando para una mayor eficiencia y mejores resultados del algoritmo, pasan por eliminar las entradas antiguas del diccionario.

Un aspecto que se repite en el entorno dinámico, es el incremento del tiempo de procesamiento del algoritmo KRLS debido a la masiva inclusión de elementos en el diccionario. Al igual que ocurria en el otro esquema, ante esta señal, suponen tiempos asumibles, pero en el caso escalón medio nos ha presentado mayores problemas.

Aparte de estos comentarios, la situación de variación para el canal escalón suave es semejante al mostrado en el entorno dinámico, por lo que tendremos como mejor opción el algoritmo KNN seguido a distancia de 2 décadas de RBF. En este caso, el algoritmo LMS tendría sentido como elección optima hasta la cota aproximada de 8 dB, reduciendo por tanto el punto de separación de los algoritmos 2 dB.

Por último, y para corroborar los resultados obtenidos con la señal escalón suave, introducimos en el sistema la señal escalón medio, obteniendo los resultados que mostramos en la Figura 5.31 para la gráfica de SER, en la Figura 5.35 para la gráfica de patrones y por último en la Tabla 5.19 para las gráficas de error de los algoritmos.

Un aspecto a tener en cuenta, y que ya apareció en el entorno dinámico y guiado por decisión con señal escalón suave, es el problema de las inserciones en el diccionario descontroladas por parte del algoritmo KRLS. En esta ocasión, de igual manera que pasó en el entorno dinámico, encontramos que el tiempo de procesamiento crece de manera insostenible, lo que nos obligó a reducir el número de muestras de simulación para la gráfica de SER frente a SNR.

Aparte de este apunte, volvemos a experimentar diferencias significativas con el entorno dinámico, ya que observando el resultado de patrones, tenemos

que para el entorno guiado por decisión, sólo tenemos un algoritmo que devuelve un valor estable, que es KNN. En esta ocasión perdemos las alternativas de LMS y KRLS que aparecían en el entorno dinámico, ya que en este caso van hacia la equiprobabilidad con motivo de la propagación de errores que sufren, y que puede verse en las gráficas de error asociadas para una SNR de 15 dB.

Como no podría ser de otra forma, solo tenemos la alternativa de KNN que devuelve un resultado viable ante la modificación de los coeficientes con una señal escalón medio.

Se obtuvieron igualmente resultados de la eficiencia de los algoritmos ante las diferentes señales de variación, que se mostraron en las tablas Tabla 5.20, Tabla 5.21, Tabla 5.22 y Tabla 5.23 para una SNR de 0 dB (se eligió este punto por ser el peor caso de todo el barrido de SNR. Esta afirmación se justificó en la Figura 5.36). En ellas volvimos a comprobar el mismo comportamiento encontrado en el entorno dinámico, en el que el algoritmo KRLS presenta un tiempo de procesamiento muy elevado, y con mucha variabilidad en función de la señal de variación introducida. Este resultado junto con el presente en la gráfica de tamaño de la máquina de KRLS de la Tabla 5.24, nos justifica de manera conveniente los problemas que encontramos con la señal de variación escalón medio en las gráficas de SER frente a SNR, derivados principalmente del incremento desproporcionado del número de elementos en el diccionario del KRLS. Un control sobre dichos elementos reduciría la tasa de reducción del diccionario y en consecuencia los tiempos de proceso.

Examinando el resultado de eficiencia del resto de algoritmos, tenemos que los más estables y con menor tiempo de proceso siguen siendo las alternativas RBF, KNN y Volterra, siendo en cualquier caso la alternativa menos costosa el algoritmo LMS.

Como conclusión del entrenamiento dinámico guiado por decisión, tenemos que al igual que ocurría en el entorno dinámico, no podemos dar un algoritmo como claro vencedor en nuestra comparativa, ya que KRLS presenta buenas prestaciones en seguimiento, pero malos resultados en cambios bruscos de canal y horribles en eficiencia (ante situaciones extremas como SNR demasiado bajas o señales de variación muy elevada). Por otro lado, tenemos el

algoritmo RBF, que presenta buen comportamiento de enganche y eficiencia, pero no en cambios bruscos de canal, y por último KNN no presenta buenos resultados en enganche pero en contra es el único algoritmo que encuentra una solución estable ante cambios bruscos de canal, con una buena eficiencia.

## *6.4 Conclusiones*

Como conclusiones generales del proyecto fin de carrera, tenemos que KRLS presenta una buena respuesta bajo entorno de canal estáticos no lineales, semejante a la que pueden ofrecer algoritmos basados en redes RBF o en agrupamientos KNN. Igualmente presenta unos buenos resultados en entornos dinámicos con señales de variación lenta, estudiados sobre problemas de seguimiento de señal, igualando los resultados de alternativas como el algoritmo RBF expuesto. También hemos obtenido muy buenos resultados, en comparación, bajo entorno dinámicos con señales de variación lenta en esquemas de entrenamiento guiados por decisión, igualando de nuevo a la alternativa de RBF. Otro aspecto positivo es el comportamiento del algoritmo ante cambios bruscos de canal no lineal, ya que en esta ocasión se mantiene muy cercano al resultado ofrecido por la mejor alternativa, KNN.

A pesar de todos estos puntos a favor, hemos encontrado un gran problema ante una situación de cambio brusco de canal no lineal en un esquema de entrenamiento guiado por decisión, ya que el algoritmo KRLS no converge, devolviendo resultados equiprobables. La explicación más razonable que intenta dar sentido a este comportamiento es que ante un cambio brusco de canal, el algoritmo mantiene las antiguas muestras pertenecientes al anterior estado en el diccionario, incluyendo a su vez nuevas para intentar establecer un conjunto de vectores que represente al nuevo estado del canal. Esas entradas inservibles en el diccionario, hacen que se dispare el número de inserciones nuevas, ya que el algoritmo no es capaz de representar las nuevas entradas con esos vectores almacenados. La consecuencia de estas inserciones es un incremento en el tiempo de proceso del algoritmo de manera lineal con el número de muestras introducidas que es visible tanto en entornos dinámicos como guiados por decisión. En estos últimos es donde encontramos mayores problemas, al tener que estimar nuestros datos de entrada al algoritmo, lo cual provoca un efecto de

propagación de errores en el entrenamiento que tiende a que el algoritmo KRLS diverga. Este problema, se encuentra presente no obstante en todas las señales de variación, aunque es en el caso más extremo en el que se hace más patente.

Aparte de este punto en contra, hemos encontrado otro especialmente importante, relacionado con la eficiencia del algoritmo, ya que en todos los esquemas de entrenamiento que hemos implementado, el algoritmo KRLS ha obtenido un tiempo de proceso mucho mayor que el resto de las alternativas. El problema principal vuelve a radicar en el incremento indiscriminado de muestras en el diccionario ante cambios de canal.

Una posible solución que planteamos para ambos problemas ante este tipo de escenarios, es el olvido controlado de entradas del diccionario que no sirvan para mapear las nuevas entradas del sistema generadas con el nuevo estado del canal. Si procedieramos de esta forma, tendríamos una complejidad menor, controlando la tasa de reducción del algoritmo (entendida como el número de elementos almacenados en el diccionario en función del número de muestras del entrenamiento), evitando de esa manera que el algoritmo diverga en entornos de cambio brusco de los coeficientes de canal en entrenamientos guiados por decisión.

Podríamos pensar que otra manera de solucionar el problema podría encontrarse en el control sobre el parámetro que gobierna el test ALD del algoritmo (este test es el encargado de establecer si la estimación realizada de la nueva entrada con los elementos del diccionario es suficientemente buena, o es necesaria la inclusión de una nueva entrada en el mismo), determinado por el parámetro $\nu$. Podríamos aumentar dicho parámetro con lo que conseguiríamos que se introdujeran menos muestras en nuestro diccionario, pero este hecho derivaría en un aumento de la tasa de error de símbolo (SER) proporcionada por el algoritmo, lo cual es un aspecto muy poco deseable.

Aparte de estos aspectos negativos comentados, tenemos otros positivos que presenta, como una tasa de error cuadrático medio muy reducida en media y varianza en comparación con el resto de algoritmos incluidos en la comparación y un tiempo de convergencia en entrenamiento muy reducido, de aproximadamente 100 muestras a lo sumo. Esta última característica es obtenida en parte debido a

la mayor complejidad del algoritmo medida en tiempo de procesamiento necesario para el entrenamiento.

Esta mayor complejidad ha sido determinante en los resultados del proyecto, ya que en igualdad de resultados con la alternativa basada en RBF (en señales de variación lenta para seguimiento) o con KNN (en señales que provocan un cambio brusco de canal para entornos dinámicos no guiados por decisión) ha salido perdiendo KRLS por su mayor tiempo de proceso relativo. Un mayor tiempo de proceso no deseable (y en ocasiones no asumible), ya que nuestro objetivo era un marco de trabajo online, y nos haría recaer en retardos no esperados.

Recapitulando acerca del resto de algoritmos utilizados en la comparativa realizada para medir las capacidades del algoritmo KRLS, podemos extraer algunas conclusiones referentes a cada uno de ellos.

Con respecto al algoritmo desarrollado sobre los kernels de Volterra y la teoría del punto fijo, hemos visto que ha presentado graves problemas de convergencia durante todos los experimentos realizados en el proyecto. A su vez comprobamos como variaban los parámetros óptimos (en terminos de minimizar la SER obtenida con el algoritmo) en función de la señal de variación introducida a los coeficientes del canal, permitiendo una estrecha franja de valores sobre los que el algoritmo obtiene una solución estable (explicamos este efecto como una consecuencia de la formulación del algoritmo basada en la teoría del punto fijo, y mediante la cual se debe disponer de un mapeo de contracción para alcanzar una solución correcta). Igualmente hemos visto que es el algoritmo de todos los estudiados con el mayor error cuadrático medio, y el que presenta mayores problemas cuando se cambia el esquema de entrenamiento estático a guiado por decisión. No obstante ofrece unos tiempos de proceso muy reducidos y constantes, lo cual es muy favorable. Por todos estos motivos consideramos el algoritmo de Volterra como el peor de todos los algoritmos estudiados en la comparativa.

Por otro lado, tenemos la alternativa implementada sobre el algoritmo clasificador Knn, que presenta una buena eficiencia en comparación con el resto elementos estudiados. Dicha eficiencia se traduce en un fácil proceso de entrenamiento, y un proceso de test más costoso, pero dependiente del tamaño de la ventana de muestras de nuestro mapa Knn. A efectos prácticos el tiempo de

procesamiento en test es muy parecido al ofrecido por RBF y Volterra. En su implementación encontramos principalmente dos problemas: el primero de ellos corresponde con la necesidad de truncar el tamaño del mapa de muestras etiquetadas del algoritmo. Mediante este mapa realizamos el proceso de búsqueda de los K vecinos más cercanos, por lo que es preciso disponer siempre de un mapa actualizado y acotado para poder tener control sobre la SER obtenida, así como de la complejidad y la memoria del canal (recordemos que la memoria era uno de los problemas detectados en el algoritmo KRLS. El comportamiento del algoritmo KNN, con un mapa enventanado, en entornos de cambio brusco de canal no hace más que justificar nuestra hipótesis acerca del causante del error en ese escenario para KRLS). El segundo de los problemas corresponde con la mala respuesta que presenta el algoritmo ante señales de variación lenta destinadas a probar la capacidad de seguimiento. Remitiendonos a los resultados prácticos obtenidos, tenemos que en esos escenarios, el algoritmo KNN presenta peores prestaciones que las ofrecidas por la alternativa lineal. A pesar de este aspecto, tenemos un punto muy positivo, ya que el algoritmo igualador no lineal basado en agrupamientos Knn, ofrece la mejor alternativa en entornos de cambio brusco de canal, pasando a ser la única alternativa de todas las mostradas en esquemas de entrenamiento guiado por decisión.

El siguiente algoritmo que hemos tenido en cuenta es el basado en redes neuronales RBF. Con él conseguimos obtener la mejor alternativa en escenarios dinámicos de variación lenta, mejorando el resultado de KRLS por motivos de eficiencia. Por el contrario presenta peores prestaciones que KRLS en entornos de cambio brusco de canal para entorno no guiados por decisión (en entornos guiados por decisión tenemos que mejora el resultado de KRLS, ya que éste diverge). Hemos encontrado muchos aspectos positivos del algoritmo, como pueden ser una mayor eficiencia y estabilidad en el tiempo de proceso con independencia de la señal de variación aplicada, o un error cuadrático medio muy reducido y semejante al mostrado por KRLS tanto en media como en varianza. No obstante, la menor complejidad del algoritmo, deriva en un tiempo de convergencia algo mayor que KRLS, lo cual nos hace necesitar un número de muestras mayor de entrenamiento para que el algoritmo basado en RBF encuentre una solución correcta y se sintonice con ella (aproximadamente 400 muestras en entornos estáticos).

Por último tenemos la alternativa lineal LMS incluida en el estudio, la cual ha sido una sorpresa a efectos prácticos, ya que a excepción del escenario con cambios bruscos de canal y esquema de aprendizaje guiado por decisión en el que divergía, hemos obtenido siempre un gran comportamiento por su parte para SNR bajas. El punto general de ruptura en el que el resultado de los algoritmos no lineales se despegaba del obtenido con LMS se podría considerar aproximadamente como 10 dB. Para todas las SNR menores a esa cota, tenemos que la alternativa LMS es la mejor opción ya que es la menos costosa y en definitiva la más eficiente. Un problema que nos podría surgir es que esa cota de ruptura se modificase, ya que es dependiente del canal no lineal usado como ejemplo en nuestras simulaciones (lo tomamos del documento de referencia de los autores de KRLS), y de la posibilidad de separar las nubes de puntos de salida del canal con una frontera de decisión lineal en presencia de mayor ruido (una menor SNR se traduce en una mayor presencia de ruido en las nubes de puntos de la salida del canal)), por lo que no podríamos basarnos exactamente de esa cota, llegando incluso a no poder ser una solución aceptada ante determinados canales. Un aspecto que se observa una vez superada la cota de ruptura, es que el algoritmo incurre en un error cuadrático medio mayor que el resto de algoritmos, con un tiempo de convergencia muy elevado, obteniendo en consecuencia una SER mayor.

A modo de conclusión hemos recorrido las posibles alternativas de aprendizaje sobre un canal de ejemplo, realizando comparaciones de prestaciones entre nuestro algoritmo bajo estudio, Kernel Recursive Least Squares (KRLS), y un conjunto de algoritmos elegidos dentro de los campos tomados como estado del arte en igualación de canal no lineal adaptativa online. El resultado de esa comparación es que el algoritmo KRLS obtiene buenas prestaciones y lo podríamos considerar como el algoritmo que mejor se ha adaptado a todos los cambios de señal y de entrenamiento que hemos realizado, aunque tiene ciertas deficiencias en su definición de diccionario a nuestro parecer que lo hacen llevar a un tiempo de proceso más elevado (en ocasiones inconsistente). No obstante hemos propuesto alternativas para intentar solucionar esos aspectos negativos.

# Capítulo 7 Apéndices

## *7.1 Apéndice A*

En este apendice mostraremos la información contenida en la referencia de los autores a su implementación de KRLS, y en la cual hemos basado la nuestra. La información puede ser encontrada en:

http://www.cs.ualberta.ca/~yaki/krls_doc.html.

Desde esta fuente puede ser descargado igualmente el código fuente desarrollado en c por los autores.

Los pasos para poder ejecutar el código de los autores se encuentran detallados igualmente en la página web, así como los ejemplos de funcionamiento que hemos utilizado para corroborar el correcto funcionamiento de nuestra implementación base de KRLS que mostramos en 5.2.1.

# Capítulo 8 Bibliografía


[Agarossi, Bellini, Bregoli, Migliorati, 1998]

Agarossi, L., Bellini, S., Bregoli, F., Migliorati, P. (1998). Equalization of non-linear optical channels. IEEE Intern. Communications. 2:662 – 667.

[Assaf, Assad, Harkouss, 2006]

Assaf, R., Assad, S. El, Harkouss, Y. (2006). Adaptive Equalization of Nonlinear Time Varying-Channels using Radial Basis Network. Information and Communication Technologies. ICTTA ’06. 1:1866 – 1871.

[Bang, Sheu, 1992]

Bang, S. H., Sheu, B. J. (1992). A neural-based digital communication receiver for intersymbol interference and white Gaussian noise channel. IEEE Proc. Int. Symp. Circuits Syst. 2933 – 2936.

[Bouchired, Ibnkahla, Roviras, Castanie, 1998]

Bouchired, S., Ibnkahla, M., Roviras, D., Castanie, F. (1998). Equalization of satellite UMTS channels using RBF networks. IEEE Inter. Symp. On Personal, Indoor and Mobile Radio Communications. 3:1250 – 1254.

[Cha, Kassam, 1995]

Cha, Z., Kassam, S. A. (1995). Channel Equalization Using Adaptive Complex Radial Basis Function Networks. IEEE Journal on Selected Areas in Communication. 13:122 – 13.

[Chen, Mulgrew, Grant, 1993]

Chen, S., Mulgrew, B., Grant, P. M. (1993). A Clustering Technique for Digital Communications Channel Equalization Using Radial Basis Function Networks. IEEE Trans. On Neural Networks. 4:570 – 578.

[Chithra, Aswathy, Sagar, 2007]

Chithra Raj, N., Aswathy, P. V., Sagar, K. V. (2007). Determination of Angle of Arrival using Nonlinear Support Vector Machine Regressors. ICSCN '07 On Signal Processing, Communications and Networking. 512 – 515.

[Cover, 1965]

Cover T. M. (1965). Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. IEEE Trans. on Electronic Computers. 14:326 – 334.

[Duda, Hart, 1973]

Duda, R. O., Hart, P. E. (1973). Pattern classification. A Wiley-Interscience Publication, New York: Wiley.

[Engel, Mannor, Meir, 2004]

Engel, Y., Mannor, S., Meir, R. (2004). The Kernel Recursive Least-Squares algorithm. IEEE Trans. on Signal Processing. 52:2275 – 2285.

[Fang, Jiao, Zhang, Pan, 2001]

Fang, Y.-W., Jiao, L.-Ch., Zhang, X.-D., Pan, J. (2001). On the convergence of Volterra filter equalizers using a pth-order inverse approach. IEEE trans. on Signal Processing. 49:1734 – 1744.

[Gibson, Siu, Cowan, 1989]

Gibson, G. J., Siu, S., Cowan, C. F. N. (1989). Multilayer perceptron structures applied to adaptive equalizers for data communications. IEEE Proc. ICASSP Glasgow, Scotland, 1183 – 1186.

[Haykin, 2005]

Haykin, S. (2005). Neural Networks and Learning Machines. Prentice Hall.

[Ibnkahla, Castanie, 1995]

Ibnkahla, M., Castanie, F. (1995). Vector neural Networks for digital satellite communications. IEEE Inter. Conf. On Communications. 3:1865 – 1869.

[Junxia, Liping, Jingming, Hua, 2004]

Junxia, T., Liping, D., Jingming, K., Hua, W. (2004). Online adaptive nonlinear channel equalization using RBF neural networks. Proc. Computational Electromagnetics and Its Applications. 300 – 303.

[Kalman, 1960]

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. Trans. ASME J. Basic Eng., 82:34-35.

[Kohonen, 1982]

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics. 43:59 – 69.

[Kumar, Saratchandran, Sundararajan, (1998)]

Kumar, P. C., Saratchandran, P., Sundararajan, N. (1998). Communications channel equalisation using minimal radial basis function neural networks. IEEE Proc. On Neural Networks for Signal Processing. 6:477 – 485.

[Kumar, Saratchandran, Sundararajan, (2000)]

Kumar, P. C., Saratchandran, P., Sundararajan, N. (2000). Minimal radial basis function neural networks for nonlinear channel equalisation. IEEE Proc. On Vision, Image and Signal Processing. 147:428 – 435.

[Haykin, 2002]

Haykin, S. (2002). Adaptive Filter Theory. Prentice – Hall, Upper Saddle River, NJ, 4th edition.

[Lee, 1996]

Lee, P. (1996). Neural-net equalization for a magnetic recording channel. Proc. 27th Asilomar Conf. Signals, Syst., Comput. 369 – 374.

[Lee, Beach, Tepedelenlioglu, 1999]

Lee, P., Beach, C., Tepedelenlioglu, N. (1999). A practical Radial Basis Function Equalizer. IEEE Trans. On Neural Networks. 10:450 – 455.

[Lin, Unbehauen, 1992]

Lin, J. N., Unbehauen, R. (1992). 2-D adaptive Volterra filter for 2-D nonlinear channel equalisation and image restoration. Electronic Letters. 28:180 – 182.

[Lincoll, 1986]

Lincoll, D. D. (1986). Learning internal representations by error propagation in Parrallel Distributed Processing: Explorations in the Microstructure of Cognition. Rumelhart, D. E., McClelland, J. L.. Eds. Cambridge, MIT Press, 318 – 362.

[Lucky et al., 1968]

Lucky, R. W., Salz, J., and Weldon, E. J. (1968). Principles of Data Communication. McGraw – Hill, New York, NY.

[Meghriche, Femmam, Derras, Haddadi, 2001]

Meghriche, K., Femmam, S., Derras, B., Haddadi, M. (2001). A non linear adaptive filter for digital data communication. Sixth Inter. Symp. on Signal Processing and its Applications. 1:304 – 306.

[Mimura, Hamada, Furukawa, 2002]

Mimura, M., Hamada, N., Furukawa, T. (2002). A recurrent RBF network for non-linear channel with time-varying characteristic. IEEE Proc. Acoustics, Speech, and Signal Processing. 1: 1081 – 1084.

[Nair, Moon, 1994]

Nair, S. K., Moon, J. (1994). Nonlinear equalization for data storage channels. Proc. IEEE Int. Conf. Commun., New Orleans, LA.

[Nair, Moon, 1995]

Nair, S. K., Moon, J. (1995). Simplified nonlinear equalizers. IEEE Trans. Magn.. 24:3051 – 3053.

[Nair, Moon, 1997]

Nair, S. K., Moon, J. (1997). A theoretical study of linear and nonlinear equalization in nonlinear magnetic storage channels. IEEE Trans. on Neural Networks. 8:1106 – 1118.

[Nicolae, Botoca, Budura, 2004]

Nicolae, M., Botoca, C., Budura, G. (2004). Nonlinear complex channel equalization using a radial basis function neural network. Neural Network Applications in Electrical Engineering. 73 – 78.

[Nowak, Van Venn, 1997]

Nowak, R. D., Van Veen, B. D. (1997). Volterra filter equalization: a fixed approach. IEEE Trans. On Signal Processing. 45: 377 – 388.

[Oppenheim, 1996]

Oppenheim, A. V. and Willsky A. S. (1996). Signal and Systems. Prentice – Hall, Englewood Cliffs, NJ.

[Oppenheim, 1996]

Oppenheim, A. V. and Willsky A. S. (1996). Signal and Systems. Prentice – Hall, Englewood Cliffs, NJ.

[Ozden, Kayran, Panayirci, 1998]

Ozden, M. T., Kayran, A. H., Panayirci, E. (1998). Adaptive Volterra channel equalisation with lattice orthogonalisation. IEEE Proc. Communications. 145:109 – 115.

[Paquier, Ibnkahla. 1998]

Paquier, W., Ibnkahla, M. (1998). Self-organizing maps for rapidly fading nonlinear channel equalization. IEEE Neural Networks Proc.. 2:865 – 869.

[Proakis, 2001]

Proakis, J. G. (2001). Digital Communications. McGraw – Hill, New York, NY.

[Raheli, Polydros, Tzon, 1994]

Raheli R., Polydros A., Tzon C.K.. PSP: A general approach to MLSE in uncertain-environments. IEEE Transactions on Communications, 13:354 – 364.

[Ramamoorthy, Keskinoz, Kumar, 2005]

Ramamoorthy, L., Keskinoz, M. Kumar, B. V. K. V. A. (2005). Support vector machines based data detection for holographic data storage systems. IEEE Inter. Conf. on Acoustics, Speech, and Signal Processing. 3:969 – 972.

[Redfern, Zhou, 1998]

Redfern, A. J., Zhou, G. T. (1998). A root method for Volterra system equalization. IEEE Signal Processing Letters. 5:285 – 288.

[Savazzi, Favalli, Costamagna, Mecocci, 1998]

Savazzi, P., Favalli, L., Costamagna, E., Mecocci, A. (1998). A suboptimal approach to channel equalization based on the nearest neighbour rule. IEEE Journal on Selected Areas in Communications. 16:1640 – 1648.

[Sebald, Bucklew, 2000]

Sebald, D. J., Bucklew, J. A. (2000). Support vector machine techniques for nonlinear equalization. IEEE Trans. On Acoustics, Speech, and Signal Processing. 48: 3217 – 3226.

[Seshadr, 1994]

Seshadr, N. (1994). Joint data and channel estimation using blind Trellis search techniques. IEEE Transactions on Communications, 2:1000 – 1001.

[Siu, Gibson, Cowan, 1990]

Siu, S., Gibson, G. J., Cowan, C. F. N. (1990). Decision feedback equalization using neural-network structures and performance comparison with standard architecture. Inst. Electron. Eng. Proc., 37:221 – 225.

[Sklar, 1988]

Sklar B. (1988). Digital Communications: Fundamentals and Applications. Prentice – Hall, Englewood Cliffs, NJ.

[TDS, 2005]

Teoría de la asignatura “Tratamiento Digital de Señales”. Junio 2005.

[Tseng, Powers, 1993]

Tseng, S.-H., Powers, E. J. (1993). Nonlinear channel equalization in digital satellite systems. IEEE Global Telecommunications Conference. 3:1639 – 1643.

[Veciana, Zakhor, 1992]

Veciana, G. de, Zakhor, A. (1992). Neural net-based continuous phase modulation receivers. IEEE Trans. Commun., 40:1396 – 1408.

[Viterbi, 1967]

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. on Information Theory, 13:260 – 269.

[Wang, Hua Lin, Zhang, Sekiya, Yahagi, 2002]

Wang, X., Hua Lin, J., Zhang, N., Sekiya, H., Yahagi, T. (2002). Combining RNN equalizer with SOM detector. Signal Processing International Conference. 2: 1291 – 1294.

[Wiener, 1949]

Wiener, N. (1949). Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications. MIT Press, Cambridge, MA.

[Yingwei, Sundararajan, Saratchandran, (1996)]

Yingwei, L., Sundararajan, N., Saratchandran, P. (1996). Adaptive nonlinear system identification using minimal radial basis function neural networks. IEEE ICASSP. 16:3521 – 3524.